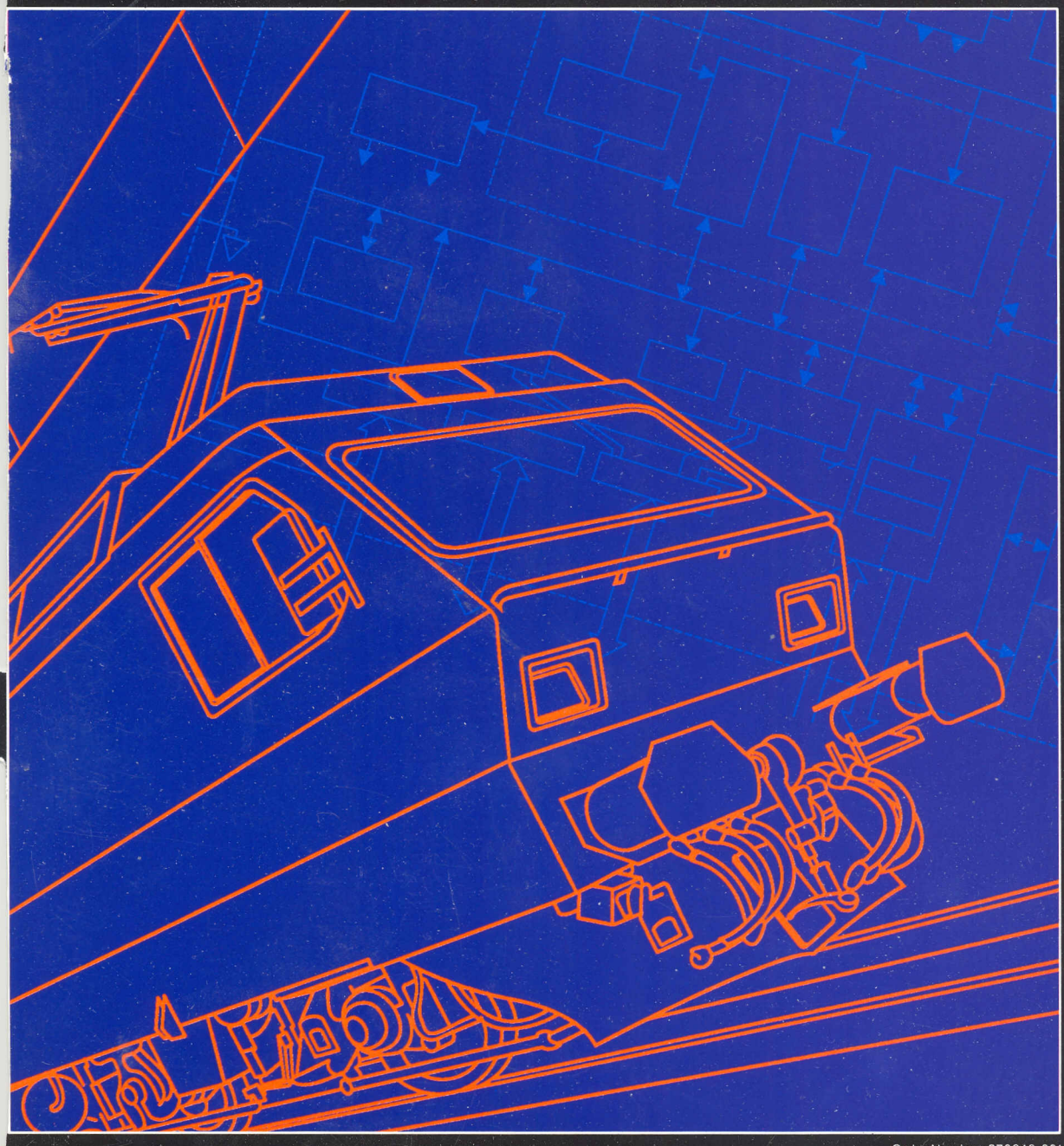
intel

16-Bit Embedded Controller Handbook



Order Number: 270646-001

intel

LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your local sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 58130
SANTA CLARA, CA 95052-8130

In the U.S. and Canada
call toll free
(800) 548-4725

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

TITLE	LITERATURE ORDER NUMBER
COMPLETE SET OF HANDBOOKS (Available in U.S. and Canada only)	231003
AUTOMOTIVE PRODUCTS HANDBOOK (Not included in handbook set)	231792
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997
EMBEDDED CONTROL APPLICATIONS HANDBOOK	270648
8-BIT EMBEDDED CONTROLLER HANDBOOK	270645
16-BIT EMBEDDED CONTROLLER HANDBOOK	270646
32-BIT EMBEDDED CONTROLLER HANDBOOK	270647
MEMORY COMPONENTS HANDBOOK	210830
MICROCOMMUNICATIONS HANDBOOK	231658
MICROCOMPUTER PROGRAMMABLE LOGIC HANDBOOK	296083
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 volume set)	230843
MILITARY PRODUCTS HANDBOOK (2 volume set. Not included in handbook set)	210461
OEM BOARDS AND SYSTEMS HANDBOOK	280407
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762
INTEL PACKAGING OUTLINES AND DIMENSIONS (Packaging types, number of leads, etc.)	231369
LITERATURE PRICE LIST (U.S. and Canada) (Comprehensive list of current Intel Literature)	210620
INTERNATIONAL LITERATURE GUIDE	E00029

CG/LIT/100188

About Our Cover:

Intel offers the most versatile high performance family of products for embedded control. Like the engine that drives the world's fastest train, Intel's embedded controllers are at the heart of these advanced designs.

Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microwave ovens, or as personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel architectures have held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, modules, and systems to give its customers the best possible competitive advantages.

16-BIT EMBEDDED CONTROLLER HANDBOOK

1989

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, ETOX, FASTPATH, Genius, i, i², ICE, iCEL, iCS, iDBP, iDIS, I²ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, Intel376, Intel386, Intel486, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, ONCE, OpenNET, OTP, PC BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE, 376, 386, 486.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

© INTEL CORPORATION 1988

intel

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide — in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Phone Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As your might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide — in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Phone Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and definitions, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; COMMENTS Magazine). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take one of our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.

Table of Contents

Alphanumeric Index ix

MCS®-96 FAMILY

Chapter 1
MCS®-96 8096BH Architectural Overview 1-1

Chapter 2
8096BH Hardware Design Information 2-1

Chapter 3
80C196KB User's Guide 3-1

Chapter 4
MCS®-96 Instruction Set 4-1

Chapter 5
Using the 8096 5-1

Chapter 6
MCS®-96 DATA SHEETS
809XBH/839XBH/879XBH with 8- or 16-Bit External Bus 6-1
809X-90, 839X-90 6-48
809XBH/839XBH/879XBH Express 6-67
809X-90, 839X-90 Express 6-76
80C196KB 16-Bit High Performance CHMOS Microcontroller 6-81
87C196KB 16-Bit High Performance CHMOS Microcontroller with 8 Kbytes of On-Chip EPROM 6-111
EV80C196KB Microcontroller Evaluation Board Fact Sheet 6-153

Chapter 7
MCS®-96 DEVELOPMENT SUPPORT TOOLS
8096 Software Development Fact Sheet 7-1
VLSiCE™ In-Circuit Emulator Fact Sheet 7-4
ICE™-196KB/PC Fact Sheet 7-7
ICE™-196KB/XX In-Circuit Emulator Fact Sheet 7-9

80186 FAMILY

Chapter 8
Using the 80186/188/C186/C188 8-1

Chapter 9
80186/188/C186/C188 DATA SHEETS
80186 High Integration 16-Bit Microprocessor 9-1
80C186 High Integration 16-Bit Microprocessor 9-57
80C187 8-Bit Numeric Processor Extension 9-121
80188 High Integration 16-Bit Microprocessor 9-151
80C188 High Integration 16-Bit Microprocessor 9-208
82188 Integrated Bus Controller for 8086, 8088, 80186, 80188 Processors 9-274
87C75PF Microcontroller Peripheral I/O Port Expander 9-291

Chapter 10
80186 DEVELOPMENT SUPPORT TOOLS
8086/80186 Software Packages 10-1
VAX/VMS* Resident 8086/8088/80186 Software Development Packages 10-5
8087 Support Library 10-13
iPAT™ Performance Tool Fact Sheet 10-17
I²ICE™ In-Circuit Emulator Fact Sheet 10-21
AEDIT Text Editor Fact Sheet 10-24
iC-86 C Compiler for 8086 10-26
ICE™-186 In-Circuit Emulator Fact Sheet 10-29
ICE™-188 In-Circuit Emulator Fact Sheet 10-33

Table of Contents (Continued)

Index

8096BH Architectural Index 11-1

Alphanumeric Index

80186 High Integration 16-Bit Microprocessor 9-1
80188 High Integration 16-Bit Microprocessor 9-151
8086/80186 Software Packages 10-1
8087 Support Library 10-13
8096 Software Development Fact Sheet 7-1
8096BH Hardware Design Information 2-1
809X-90, 839X-90 6-48
809X-90, 839X-90 Express 6-76
809XBH/839XBH/879XBH Express 6-67
809XBH/839XBH/879XBH with 8- or 16-Bit External Bus 6-1
80C186 High Integration 16-Bit Microprocessor 9-57
80C187 8-Bit Numeric Processor Extension 9-121
80C188 High Integration 16-Bit Microprocessor 9-208
80C196KB 16-Bit High Performance CHMOS Microcontroller 6-81
80C196KB User's Guide 3-1
82188 Integrated Bus Controller for 8086, 8088, 80186, 80188 Processors 9-274
87C196KB 16-Bit High Performance CHMOS Microcontroller with 8 Kbytes of On-Chip EPROM 6-111
87C75PF Microcontroller Peripheral I/O Port Expander 9-291
AEDIT Text Editor Fact Sheet 10-24
EV80C196KB Microcontroller Evaluation Board Fact Sheet 6-153
iC-86 C Compiler for 8086 10-26
ICE™-186 In-Circuit Emulator Fact Sheet 10-29
ICE™-188 In-Circuit Emulator Fact Sheet 10-33
ICE™-196KB/PC Fact Sheet 7-7
ICE™-196KB/XX In-Circuit Emulator Fact Sheet 7-9
iPAT™ Performance Tool Fact Sheet 10-17
I^2ICE™ In-Circuit Emulator Fact Sheet 10-21
MCS®-96 8096BH Architectural Overview 1-1
MCS®-96 Instruction Set 4-1
Using the 80186/188/C186/C188 8-1
Using the 8096 5-1
VAX/VMS* Resident 8086/8088/80186 Software Development Packages 10-5
VLSiCE™ In-Circuit Emulator Fact Sheet 7-4

8086/80186 Software Packages 7-1
8087 Support Library 10-1
8088 Software Development Fact Sheet 10-13
8088H Hardware Design Information 7-1
80C186 Hardware Design Information 2-1
80C187 Numeric Processor Extension 8-43
80C188 Integrated Bus Controller for 8088, 8086, 80186, 80188 Processors 6-79
80C188 Integrated Bus Controller for 8088, 8086, 80186, 80188 Processors 8-67
80C188 High Integration 16-Bit Microprocessor 6-1
80C188 High Integration 16-Bit Microprocessor 9-57
80C187 8-Bit Numeric Processor Extension 9-127
80C188 High Integration 16-Bit Microprocessor 9-209
80C186EB 16-Bit High Performance CHMOS Microcontroller 6-91
80C186EB User's Guide 3-1
82188 Integrated Bus Controller for 8088, 8086, 80186, 80188 Processors 9-274
87C196KB 16-Bit High Performance CHMOS Microcontroller with 8 Kbytes of On-Chip EPROM 8-111
87C752PF Microcontroller Peripheral I/O Port Expander 9-291
AEDIT Text Editor Fact Sheet 10-24
EV80C186KB Microcontroller Evaluation Board Fact Sheet 8-153
iC-86 C Compiler for 8086 10-28
ICE™-186 In-Circuit Emulator Fact Sheet 10-29
ICE™-188 In-Circuit Emulator Fact Sheet 10-39
ICE™-196KB/PC Fact Sheet 7-7
ICE™-196KB/XX In-Circuit Emulator Fact Sheet 7-9
iPAT™ Performance Tool Fact Sheet 10-17
iRICE™ In-Circuit Emulator Fact Sheet 10-27
MCS®-86 8086/8 Architectural Overview 1-1
MCS®-86 Instruction Set 4-1
Using the 80186/188/C186/C188 8-1
Using the 8096 6-1
VAX/VMS* Resident 8086/8088/80186 Software Development Packages 10-5
VLSICE™ In-Circuit Emulator Fact Sheet 7-4

1 8096BH Architectural Overview

MCS®-96
8096BH ARCHITECTURAL OVERVIEW

This manual is written about the 8X9XBH parts, generically referred to as an 8096BH. An 8-bit bus version of this part, the 8098/8398, is described in the 8-bit Embedded Controller Handbook.

The 8096BH can be separated into several sections for the purpose of describing its operation. There is a 16-bit CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator. The CPU and the programmable I/O make the 8096BH very different from any other microcontroller. Let us first examine the CPU.

1.0 CPU OPERATION

The major components of the CPU on the 8096BH are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

1.1 CPU Buses

A "Control Unit" and two busses connect the Register File and RALU. Figure 1 shows the CPU with its

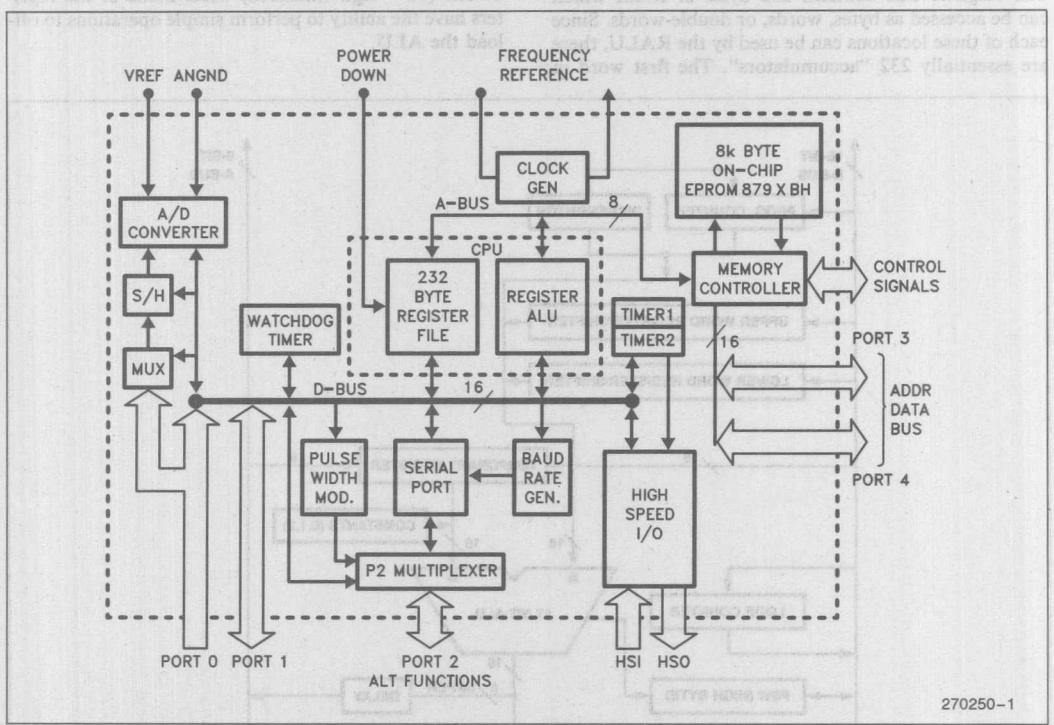


Figure 1. Block Diagram

major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

1.2 CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

1.3 RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 1 shows the instruction register and the control unit.

1.4 RALU

Most calculations performed by the 8096BH take place in the RALU. The RALU, shown in Figure 2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to offload the ALU.

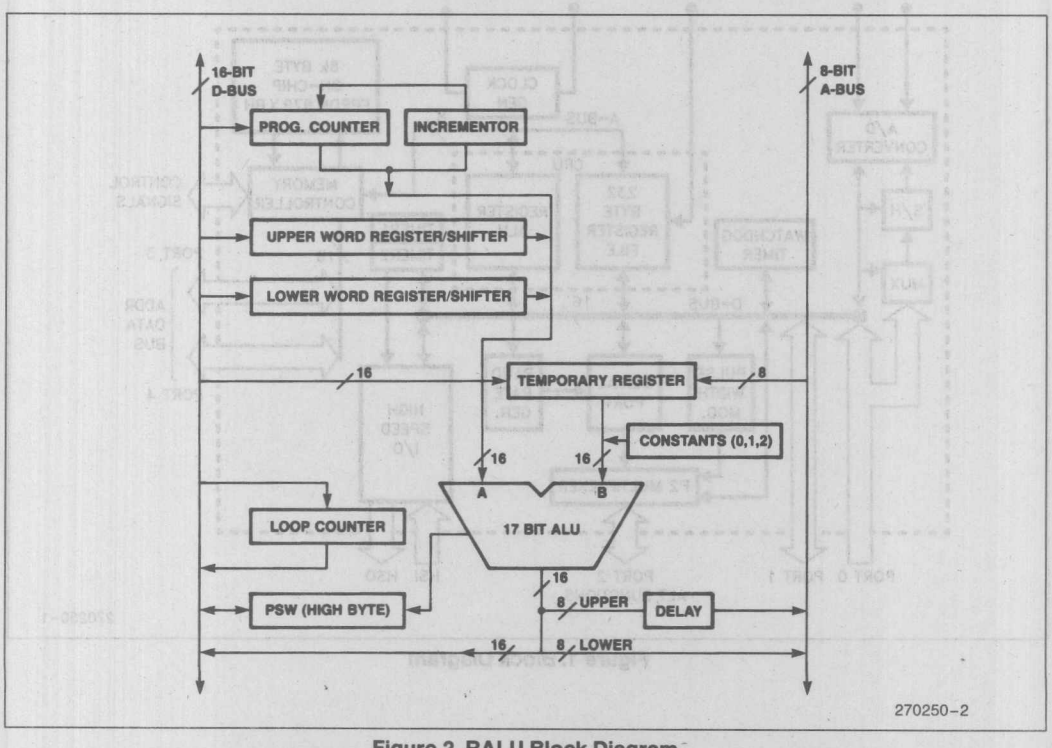


Figure 2. RALU Block Diagram

A separate incrementor is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A-Bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

1.5 Internal Timing

The 8096BH requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 3. Details of the circuit and suggestions for its use can be found in Section 1 of the Hardware Design chapter.

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096BH operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 4. It should be noted that propagation delays have not been taken into account in this diagram. Details on these and other timing relationships can be found in the Hardware Design chapter.

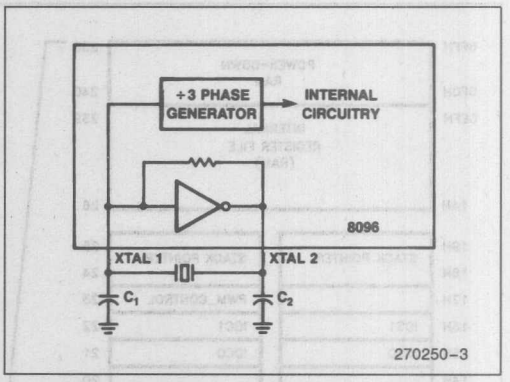


Figure 3. Block Diagram of Oscillator

The $\overline{\text{RESET}}$ line can be used to start the 8096BH at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under RESET, Section 13.

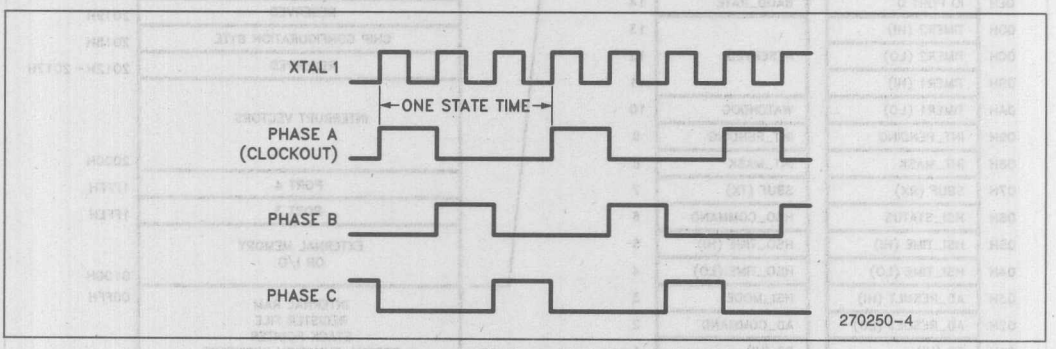


Figure 4. Internal Timings Relative to XTAL 1

2.0 MEMORY SPACE

The addressable memory space on the 8096BH consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2080H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in Figure 5.

2.1 Register File

Locations 00H through 0FFH contain the Register File and Special Function Registers, (SFRs). No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI and TRAP interrupt are is also reserved for Intel development tools.

The RALU can operate on any of the 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. These are not SFRs, and may be used as standard RAM if stack operations are not being performed. The stack pointer must be initialized by the

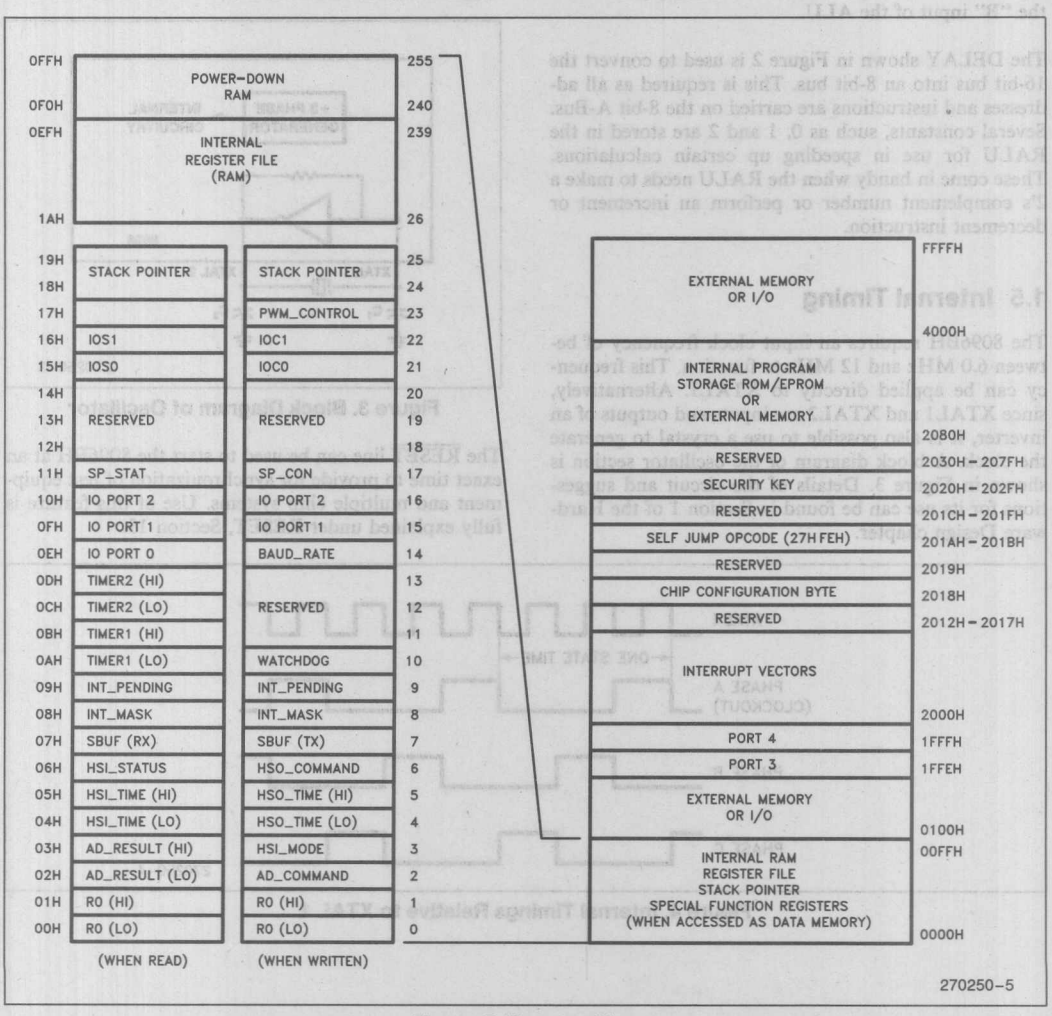


Figure 5. Memory Map

user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.

2.2 Special Function Registers

All of the I/O on the 8096BH is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 5 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 6, with complete descriptions reserved for later sections.

There are several restrictions on using special function registers.

Neither the source or destination addresses of the Multiply and Divide instructions can be a writable special function register.

These registers may not be used as base or index registers for indirect or indexed instructions.

These registers can only be accessed as bytes unless otherwise specified in Figure 6. Note that some of these registers can only be accessed as words, and not as bytes.

Within the SFR space are several registers labeled "RESERVED". These registers are reserved for future expansion and test purposes. Operations should not be performed with these registers as reads from them and writes to them may produce unexpected results. For example, in some versions of the 8096BH writing to location 0CH will set both timers to 0FFFXH. This may not be the case in future products, so it should not be used as a feature.

2.3 Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from the V_{PD} pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the V_{PD} pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification). Both V_{CC} and V_{PD} must have power applied for normal operation. If V_{PD} is not applied the power down RAM will not function properly, even if V_{CC} is applied.

To place the 8096BH into a power down mode, the $\overline{\text{RESET}}$ pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the V_{CC} pin, the V_{PD} pin must remain within specifications. The 8096BH can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096BH out of power down, $\overline{\text{RESET}}$ is held low while V_{CC} is applied. Two state times after the oscillator has stabilized, the $\overline{\text{RESET}}$ pin can be pulled high. The 8096BH will begin to execute code at location 02080H 10 state times after $\overline{\text{RESET}}$ is pulled high. Figure 7 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in the Hardware Design Chapter. Reset is discussed in Section 13.

To determine if a reset is a return from power down or a complete cold start a "key" can be written into power-down RAM while the part is running. This key can be checked on reset to determine which type of reset has occurred. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

Register	Description	Section
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3
AD_RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)	8
AD_COMMAND	A/D Command Register — Controls the A/D	8
HSI_MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	6
HSI_TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)	6
HSO_TIME	HSO Time Hi/Lo — Sets the time or count for the High Speed Output to execute the command in the Command Register. (word write only)	7
HSO_COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	7
HSI_STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins.	6
SBUF (TX)	Transmit buffer for the serial port, holds contents to be outputted.	9
SBUF (RX)	Receive buffer for the serial port, holds the byte just received by the serial port.	9
INT_MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	4
INT_PENDING	Interrupt Pending Register — Indicates that an interrupt signal has occurred on one of the sources and has not been serviced.	4
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	12
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)	5
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)	5
IOPORT0	Port 0 Register — Levels on pins of port 0.	10
BAUD_RATE	Register which determines the baud rate, this register is loaded sequentially.	9
IOPORT1	Port 1 Register — Used to read or write to Port 1.	10
IOPORT2	Port 2 Register — Used to read or write to Port 2.	10
SP_STAT	Serial Port Status — Indicates the status of the serial port.	9
SP_CON	Serial Port Control — Used to set the mode of the serial port.	9
IOS0	I/O Status Register 0 — Contains information on the HSO status	11
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	11
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	11
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	11
PWM_CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	8

Figure 6. SFR Summary

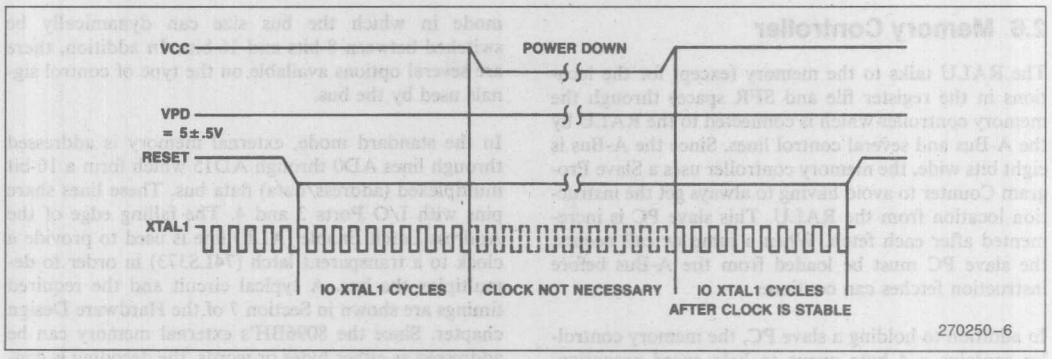


Figure 7. Power Down Timing

2.4 Reserved Memory Spaces

A listing of locations with special significance is shown in Figure 8. The locations marked "Reserved" are reserved by Intel for use in testing or future products. They must be filled with the Hex value FFH to insure compatibility with future parts.

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 7 of the Hardware Design chapter. If ports 3 and 4 are not going to be reconstructed, these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in Section 4.

Locations 2012H through 2017H are reserved for future use. Location 2018H is the Chip Configuration byte which will be discussed in the next section. The Jump-To-Self opcodes at locations 201AH and 201BH are provided for EPROM programming as detailed in the Hardware Design chapter. Locations 2020H through 202FH are the security key used with the ROM Lock feature which will be discussed in the next section. All unspecified addresses in locations 2000H through 207FH, including those marked Reserved, should be considered reserved for use by Intel.

Resetting the 8096BH causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in Section 13.

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27H FEH)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

Figure 8. Registers with Special Significance

2.5 Internal ROM and EPROM

When a ROM part is ordered, or an EPROM part is programmed, the internal memory locations 2080H through 3FFFH are user specified, as are the interrupt vectors, Chip Configuration Register and Security Key in locations 2000H through 202FH.

Instruction and data fetches from the internal ROM or EPROM occur only if the part has a ROM or EPROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory. The $\overline{EA}$ pin is latched on $\overline{RESET}$ rising. Information on programming EPROMs can be found in Section 10 of the Hardware Design chapter.

Do not execute code out of the last three locations of internal ROM/EPROM.

2.6 Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-Bus and several control lines. Since the A-Bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-Bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 4 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Section 14.8 show the normal execution times with no wait states added and the 16-bit bus selected. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in the table.

2.7 System Bus

There are several operating modes on the 8096BH. The standard bus mode uses a 16-bit multiplexed address/data bus. Other bus modes include an 8-bit mode and a mode in which the bus size can dynamically be switched between 8-bits and 16-bits. In addition, there are several options available on the type of control signals used by the bus.

In the standard mode, external memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O Ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in Section 7 of the Hardware Design chapter. Since the 8096BH's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ($\overline{BHE}$) and Address/Data Line 0 (AD0).

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

When $\overline{BHE}$ is active (low), the memory connected to the high byte of the data bus should be selected. When MA0 is low the memory connected to the low byte of

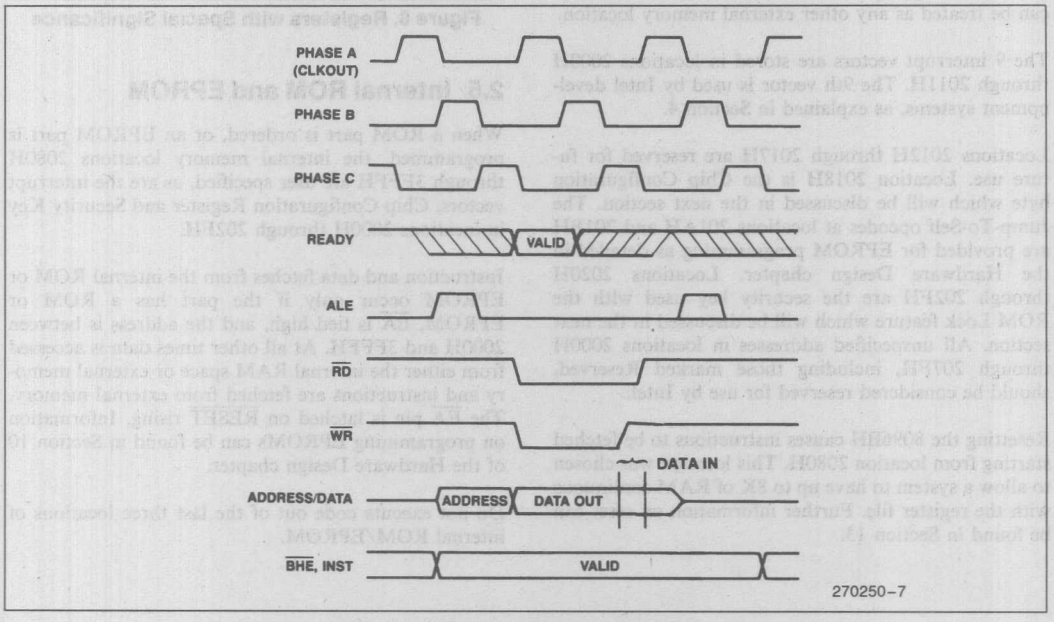


Figure 9. External Memory Timings

the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0=0, $\overline{BHE}$=1), to the high (odd) byte only (MA0=1, $\overline{BHE}$=0), or to both bytes (MA0=0, $\overline{BHE}$=0). When a memory block is being used only for reads, $\overline{BHE}$ and MA0 need not be decoded.

TIMINGS

Figure 9 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0–AD15 and $\overline{BHE}$ is set to the required state. ALE then falls, the address is taken off the pins, and the $\overline{RD}$ (Read) signal goes low. When $\overline{RD}$ falls, external memory should present its data to the 8096BH.

READ

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of $\overline{RD}$. The rising edge of $\overline{RD}$ latches the information into the 8096BH. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin. The INST pin will be low for the Chip Configuration Byte and Interrupt Vector fetches.

WRITE

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ($\overline{WR}$) signal is used instead of the $\overline{RD}$ signal. The timings are the same until the falling edge of the $\overline{WR}$ line. At this point the 8096BH removes the address and places the data on the bus. When the $\overline{WR}$ line goes high the data should be latched to the external memory. In systems which can write to byte locations, the AD0 and $\overline{BHE}$ lines must be used to decode $\overline{WR}$ into WRite to Low byte ($\overline{WRL}$) and WRite to High byte ($\overline{WRH}$) signals. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

READY

A ready line is available on the 8096BH to extend the width of the RD and WR pulses in order to allow access of slow memories or for DMA purposes. If the READY line is low by the specified time after ALE falls, the 8096BH will hold the bus lines to their values at the falling edge of CLKOUT. When the READY line rises the bus cycle will continue with the next falling edge of CLKOUT.

Since the bus is synchronized to CLKOUT, it can be held only for an integral number of state times. If more than TYLYH nanoseconds are added the processor will act unpredictably.

There are several set-up and hold times associated with the READY signal. If these timings are not met, the part may not respond with the proper number of wait states.

For falling edges of READY, sampling is done internally on the falling edge of Phase A. Since Phase A generates CLKOUT, (after some propagation delay) the sample will be taken prior to CLKOUT falling. The timing specification for this is given as TLLYV, the time between when ALE falls and READY must be valid. If READY changes between TLLYV max and the falling edge of CLKOUT (TLLYH MIN on 48-lead devices) it would be possible to have the READY signal transitioning as it is being sampled.

This situation could cause a metastable condition which could make the device operate unpredictably.

For the rising edge of READY, sampling is done internally on the rising edge of Phase A. The rising edge logic is fully synchronized, so it is not possible to cause a metastable condition once the device is in a valid not-ready condition. To cause one wait state to occur the rising edge of READY must occur before TLLYH MAX after ALE falls. If the signal is brought up after this time two wait states may occur. If two wait states are desired, READY should be brought high within the TLLYH specification + 3 Tosc. Additional wait states can be caused by adding additional state times to the READY low time. The maximum amount of time that a device may be held not-ready is specified as TYLYH.

The 8096BH has the ability to internally limit the number of wait states to 1, 2, or 3 as determined by the value in the Chip Configuration Register, (CCR). Using the CCR for ready timing is discussed at the end of this section. If a ready limit is set, the TLLYH MAX specification is not used.

OPERATING MODES

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and runtime selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register (CCR) is used to store the operating mode information.

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 10. The functions associated with each bit are described in this section.

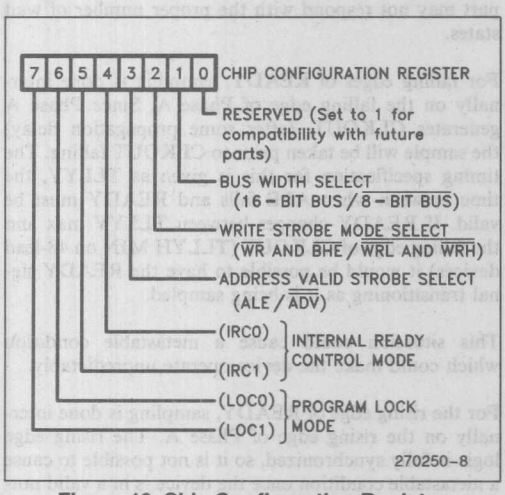


Figure 10. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

If the $\overline{EA}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{EA}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{EA}$ is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming mode is described in Section 10 of the Hardware Design chapter.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8051 style 16-bit address/8-bit data bus.

During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 11 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of Address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one.)

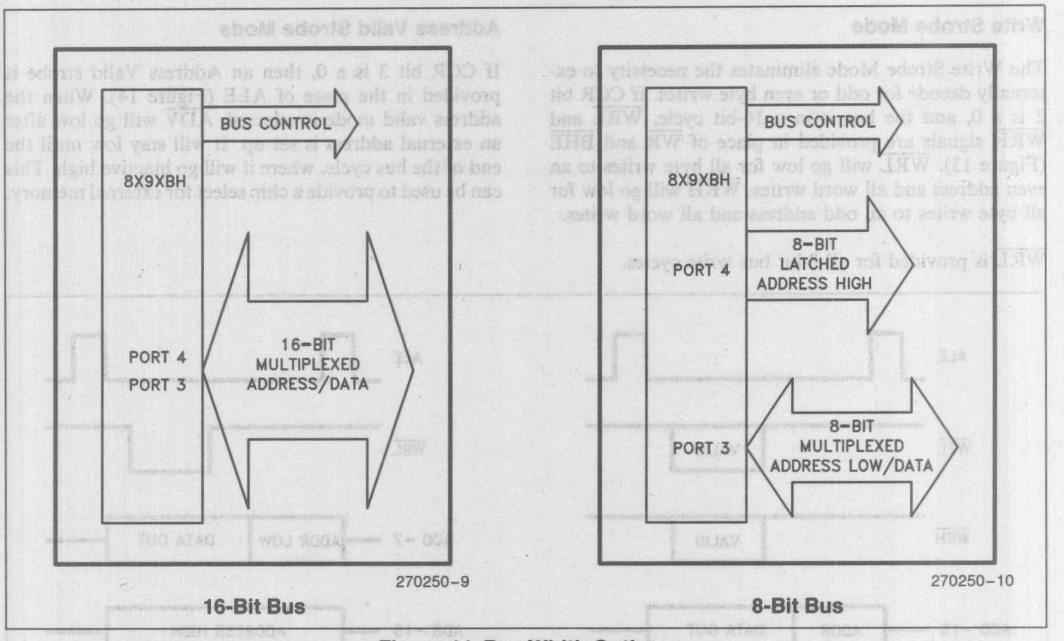


Figure 11. Bus Width Options

BUS CONTROL

Using the CCR, the 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines. Figures 12–15 show the signals which can be modified by changing bits in the CCR, all other lines will operate as shown in Figure 9.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{WR}$, $\overline{BHE}$ and ALE are provided (Figure 12). $\overline{WR}$ will come out for every write. $\overline{BHE}$ will be valid throughout the bus cycle and can be combined with $\overline{WR}$ and address line 0 to form $\overline{WRL}$ and $\overline{WRH}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address.

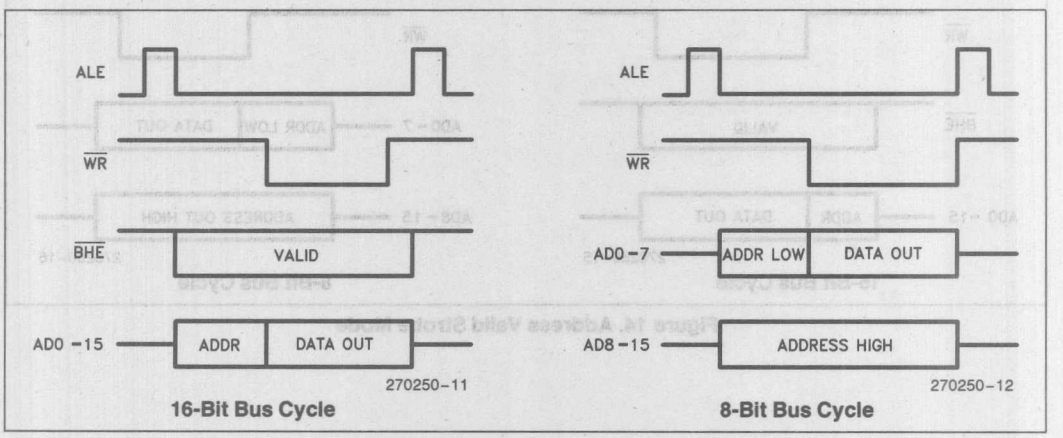


Figure 12. Standard Bus Control

Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{WRL}$ and $\overline{WRH}$ signals are provided in place of $\overline{WR}$ and $\overline{BHE}$ (Figure 13). $\overline{WRL}$ will go low for all byte writes to an even address and all word writes. $\overline{WRH}$ will go low for all byte writes to an odd address and all word writes.

$\overline{WRL}$ is provided for all 8-bit bus write cycles.

Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid strobe is provided in the place of ALE (Figure 14). When the address valid mode is selected, $\overline{ADV}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.

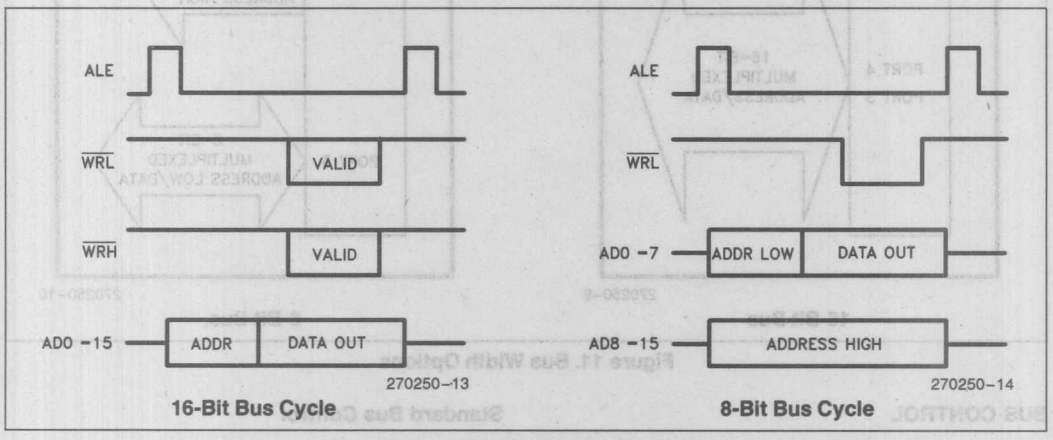


Figure 13. Write Strobe Mode

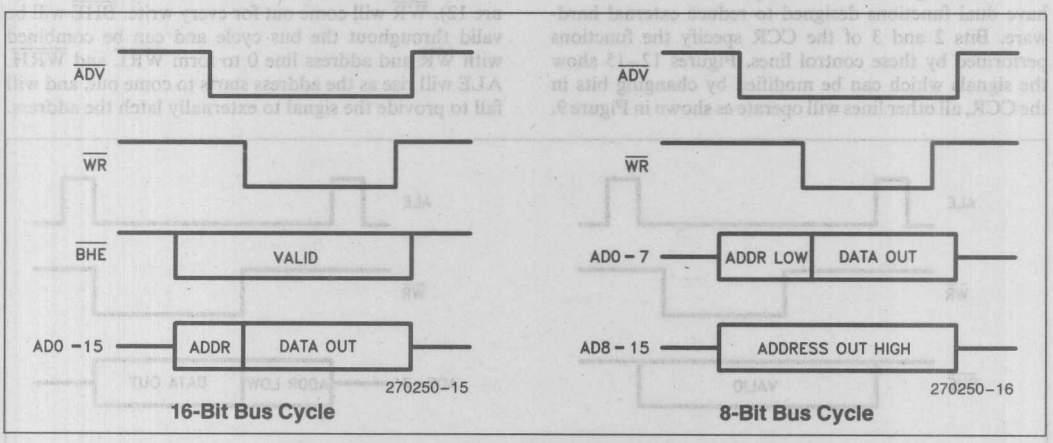


Figure 14. Address Valid Strobe Mode

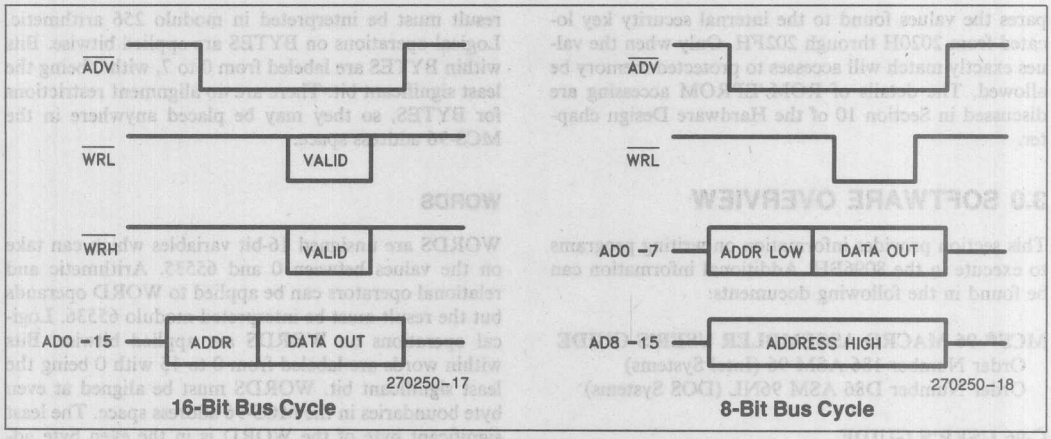


Figure 15. Write Strobe with Address Valid Strobe

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid strobe and the Write Strobes will be provided for bus control. Figure 15 shows these signals.

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal Ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the desired number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. This is because the lock protection mechanism is gated off of the Memory Controller's slave program counter and not the CPU program counter. If the bus controller receives a request to perform a read of protected memory, the read sequence occurs with indeterminate data being returned to the CPU. Note that the interrupt vectors and the CCR are not protected.

To provide verification and testing when the program lock feature is enabled, the 839XBH and 879XBH verify the security key before programming or test modes are allowed to read from protected memory. Before protected memory can be read, the chip reads external memory locations 4020H through 402FH and com-

pares the values found to the internal security key located from 2020H through 202FH. Only when the values exactly match will accesses to protected memory be allowed. The details of ROM/EPROM accessing are discussed in Section 10 of the Hardware Design chapter.

3.0 SOFTWARE OVERVIEW

This section provides information on writing programs to execute in the 8096BH. Additional information can be found in the following documents:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 186 ASM 96 (Intel Systems)
Order Number D86 ASM 96NL (DOS Systems)

C-96 USER'S GUIDE
Order Number D86 C96NL (DOS Systems)

PL/M-96 USER'S GUIDE
Order Number 186 PLM 96 (Intel Systems)
Order Number D86 PLM 96NL (DOS Systems)

Throughout this section, short sections of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16-bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in the 8086 (80186). It is important to note, however, that in the 8096BH, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within the onboard register file.

3.1 Operand Types

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion, the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

BYTES

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7, with 0 being the least significant bit. There are no alignment restrictions for BYTES, so they may be placed anywhere in the MCS-96 address space.

WORDS

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

SHORT-INTEGERS

SHORT-INTEGERS are 8-bit signed variables which can take on the values between −128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

INTEGERS

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

BITS

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

DOUBLE-WORDS

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with Intel provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in Section 3.5.

LONG-INTEGERS

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in Section 3.5.

3.2 Operand Addressing

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic address modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

REGISTER-DIRECT REFERENCES

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and register address and must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

Examples

```
ADD     AX,BX,CX      ; AX:=BX+CX
MUL     AX,BX         ; AX:=AX*BX
INCB    CL            ; CL:=CL+1
```

INDIRECT REFERENCES

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

Examples

```
LD      AX,[AX]       ; AX:=MEM_WORD(AX)
ADDB    AL,BL,[CX]    ; AL:=BL+MEM_BYTE(CX)
POP     [AX]          ; MEM_WORD(AX):=MEM_WORD(SP) ; SP:=SP+2
```

INDIRECT WITH AUTO-INCREMENT REFERENCES

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

Examples

```
LD    AX,[BX]+     ; AX:=MEM_WORD(BX) ; BX:=BX+2
ADDB  AL,BL,[CX]+  ; AL:=BL+MEM_BYTE(CX) ; CX:=CX+1
PUSH  [AX]+        ; SP:=SP-2;
                   ;   MEM_WORD(SP):=MEM_WORD(AX)
                   ;   AX:=AX+2
```

IMMEDIATE REFERENCES

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

Examples

```
ADD  AX,#340  ; AX:=AX+340
PUSH #1234H   ; SP:=SP-2; MEM_WORD(SP):=1234H
DIVB AX,#10   ; AL:=AX/10; AH:=AX MOD 10
```

SHORT-INDEXED REFERENCES

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended, the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
LD    AX,12[BX]     ; AX:=MEM_WORD(BX+12)
MULB  AX,BL,3[CX]   ; AX:=BL*MEM_BYTE(CX+3)
```

LONG-INDEXED REFERENCES

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
AND   AX,BX,TABLE[CX]    ; AX:=BX AND MEM_WORD(TABLE+CX)
ST    AX,TABLE[BX]       ; MEM_WORD(TABLE+BX):=AX
ADDB  AL,BL,LOOKUP[CX]   ; AL:=BL+MEM_BYTE(LOOKUP+CX)
```

ZERO REGISTER ADDRESSING

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

Examples

```
ADD  AX,1234[0]   ; AX:=AX+MEM_WORD(1234)
POP  5678[0]      ; MEM_WORD(5678):=MEM_WORD(SP)
                  ;  SP:=SP+2
```

STACK POINTER REGISTER ADDRESSING

The system stack pointer in the 8096BH can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

Examples

```
PUSH  [SP]      ; DUPLICATE TOP_OF_STACK
LD    AX,2[SP]  ; AX:=NEXT_TO_TOP
```

ASSEMBLY LANGUAGE ADDRESSING MODES

The 8096BH assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Indexed Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 Program Status Word

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in Figure 16. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 16. PSW Register

INTERRUPT FLAGS

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT_MASK—address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt disable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT_PENDING register even if they are locked out. Execution of the corresponding service routines will procede according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in Section 4.

CONDITION FLAGS

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

V. The V (overflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type. For the SHL, SHLB and SHLL instructions, the V flag will be set if the most significant bit of the operand changes at any time during the shift.

VT. The VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C = 0).

ST. The ST (STicky bit) flag is set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL    ;AX:=CL*DL
SHR     AX,#4       ;Shift right 4 places
```

If the C flag is set after the shift, it indicates that the bits shifted off the end of the operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift, it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the Bits Shifted Off
0 0	Value = 0
0 1	0 < Value < ½ LSB
1 0	Value = ½ LSB
1 1	Value > ½ LSB

Figure 17. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

3.4 Instruction Set

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32

by 16 divides and for shift operations. The remaining operations on 32-bit variables can be implemented by combinations of 16-bit operations. As an example the sequence:

```
ADD     AX,CX
ADDC    BX,DX
```

performs a 32-bit addition, and the sequence

```
SUB     AX,CX
SUBC    BX,DX
```

performs a 32-bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096BH (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096BH NORML instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096BH supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if unsigned operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Table 3 summarizes the operation of each of the instructions. Complete descriptions of each instruction and its timings can be found in the Instruction Set chapter. A summary of instruction opcodes and timing is included in the quick reference section at the end of this chapter. Examples of using the instruction set of the MCS-96 family can be found in the chapter, "Using the 8096", included in this handbook.

Table 3. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3. Instruction Summary (Continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	—	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	—	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↕	↕	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↕	?	↕	↕	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↕	?	↕	0	—	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↕	↕	↕	0	—	↕	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↕	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

3.5 Software Standards and Conventions

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

REGISTER UTILIZATION

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively, some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

ADDRESSING 32-BIT OPERANDS

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

SUBROUTINE LINKAGE

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16-bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE
(param1,param2,param3);
  DECLARE param1 BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

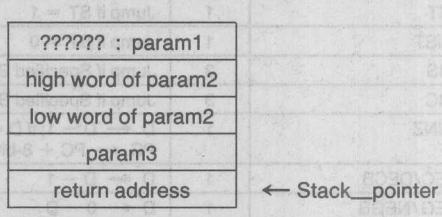


Figure 18. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8-, 16- or 32-bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a) Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b) Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c) The Program Status Word (PSW—see Section 3.3) is not saved and restored by procedures so the calling code must assume that the condition flags (Z, N, V, VT, C, and ST) are modified by the procedure.

d) Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

4.0 INTERRUPT STRUCTURE

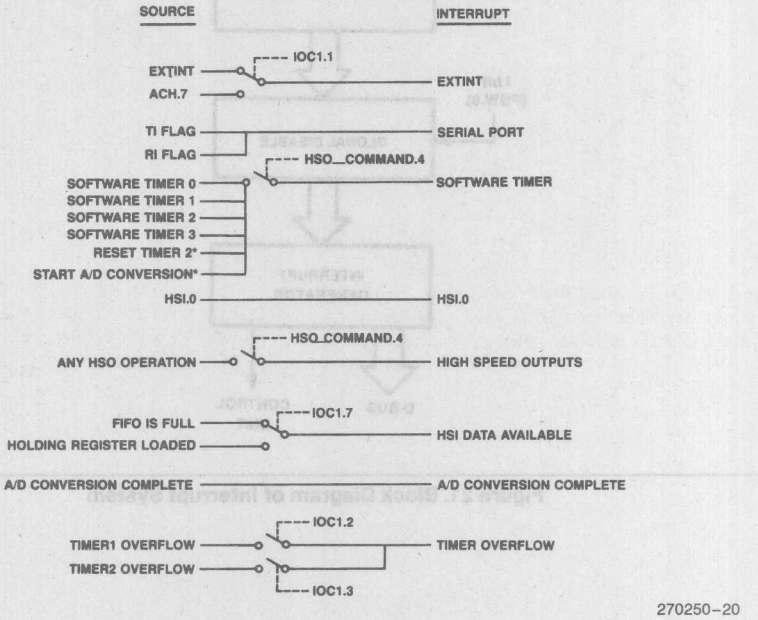
There are 21 sources of interrupts on the 8096BH. These sources are gathered into 8 interrupt types as indicated in Figure 19. The I/O control registers which control some of the sources are indicated in the figure. Each of the eight types of interrupts has its own interrupt vector as listed in Figure 20. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use in Intel development systems.

The programmer must initialize the interrupt vector table with the starting address of the appropriate interrupt service routine. It is suggested that any unused interrupts be vectored to an error handling routine. The error routine should contain recovery code that will not further corrupt an already erroneous situation. In a debug environment, it may be desirable to have the routine lock into a jump to self loop which would be easily traceable with emulation tools. More sophisticated routines may be appropriate for production code recoveries.

Three registers control the operation of the interrupt system: Interrupt Pending, Interrupt Mask, and the PSW which contains a global disable bit. A block diagram of the system is shown in Figure 21. The transition detector looks for 0 to 1 transitions on any of the sources. External sources have a maximum transition speed of one edge every state time. If this is exceeded the interrupt may not be detected.

Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 20. Interrupt Vector Locations



NOTE:
*Only when initiated by the HSO unit.

Figure 19. All Possible Interrupt Sources

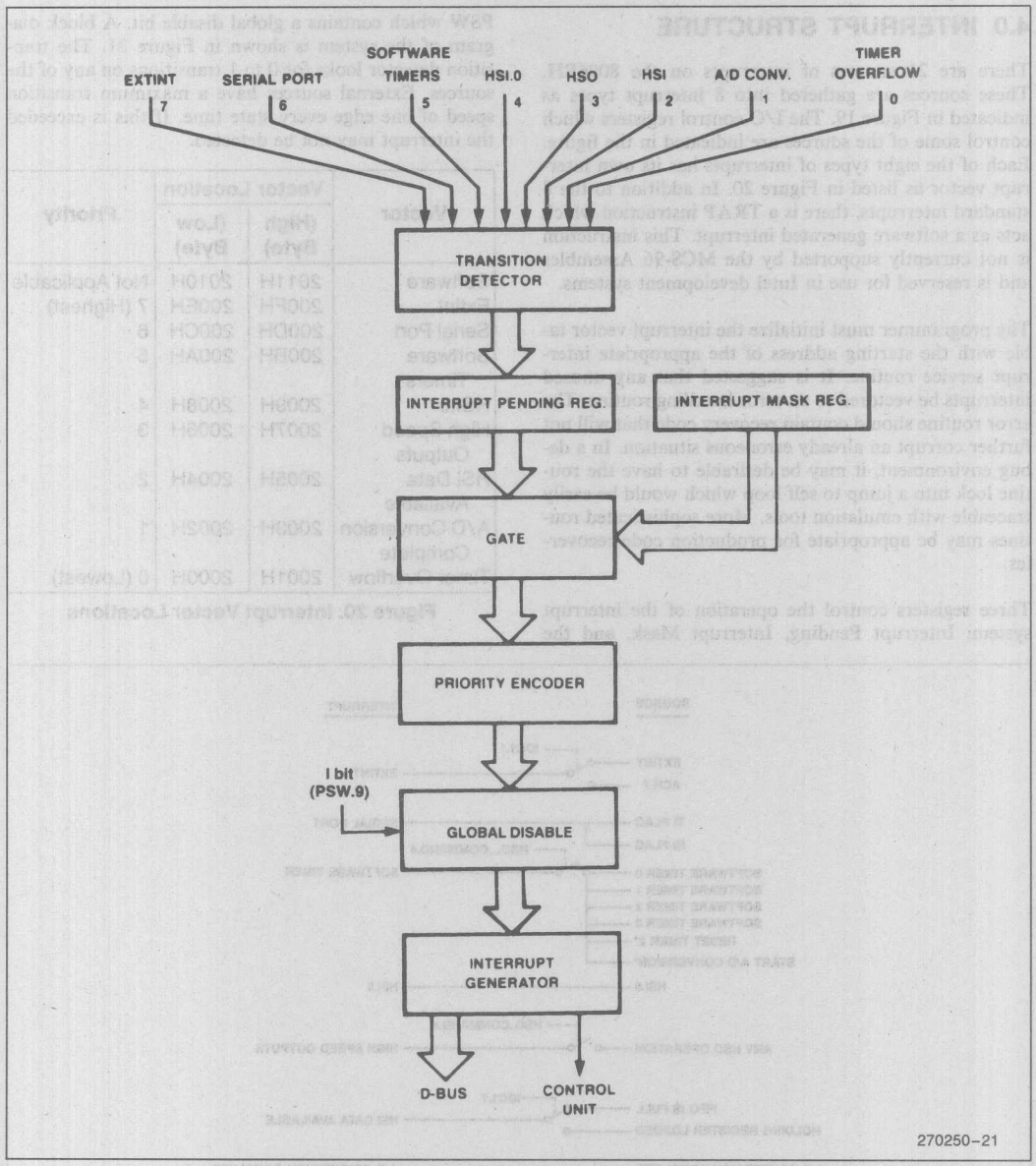


Figure 21. Block Diagram of Interrupt System

4.1 Interrupt Control

Interrupt Pending Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-09H). When the interrupt vector is taken, the pending bit is cleared. This register, the format of which is shown in Figure 22, can be read or modified as a byte register. It can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way to do this is to use the logical instructions in the two or three operand format, for example:

```
ANDB  INT_PENDING,#11111101B
          ; Clears the A/D Interrupt
ORB   INT_PENDING,#00000010B
          ; Sets the A/D Interrupt
```

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096BH will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096BH holds off acknowledging interrupts during these "read/modify/write" instructions.

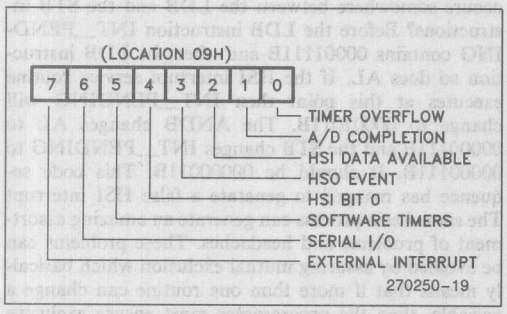


Figure 22. Interrupt Pending Register

Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-08H). The format of this register is the same as that of the Interrupt Pending Register shown in Figure 22.

The INT_MASK register can be read or written as byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The hardware will save any interrupts that occur by setting bits in the pending register, even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

GLOBAL DISABLE

The processing of all interrupts can be disabled by clearing the I bit in the PSW. Setting the I bit will enable interrupts that have mask register bits which are set. The I bit is controlled by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts. Interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

4.2 Interrupt Priorities

The priority encoder looks at all of the interrupts which are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Figure 20 (7 is highest, 0 is lowest). The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

This priority selection controls the order in which pending interrupts are passed to the software via interrupt calls. The software can then implement its own priority structure by controlling the mask register (INT_MASK). To see how this is done, consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
 PUSHF          ; Save the PSW
                  (Includes INT_MASK)
 LDB    INT_MASK,#00000100B
 EI             ; Enable interrupts again
 ;
 ;
 ;
 ;    } Service the interrupt
 ;
 ;
 ;
 POPF           ; Restore the PSW
 RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a) After the hardware decides to process an interrupt, it generates and executes a special interrupt-call instruction, which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts, the first instruction of the interrupt service routine will execute.

b) The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global disable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and, by the time the LD instruction starts, all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c) The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d) The EI instruction reenables the processing of interrupts.

e) The actual interrupt service routine executes within the priority structure established by the software.

f) At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT_MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

4.3 Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PENDING
ANDB    AL,#bit_mask
STB     AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PENDING contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PENDING to 00000111B. It should be 00000011B. This code sequence has manged to generate a false HSI interrupt The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction.

```
ANDB        INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. Changes to the INT_PENDING register must be made as a single instruction, since bits can be changed in this register even if interrupts are disabled. Depending on system configurations, several other SFRs might also need to be changed in a single instruction for the same reason.

When variables must be modified without interruption, and a single instruction can not be used, the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

4.4 Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI — Enable and Disable Interrupts

POPF, PUSHF— Pop and Push Flags

SIGND — Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide)

TRAP — Software interrupt

When an interrupt is acknowledged, the interrupt pending bit is cleared, and a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 42 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 70 (42 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled. Refer to Figure 22A, Interrupt Response Time, to visualize an example of a worst case scenario.

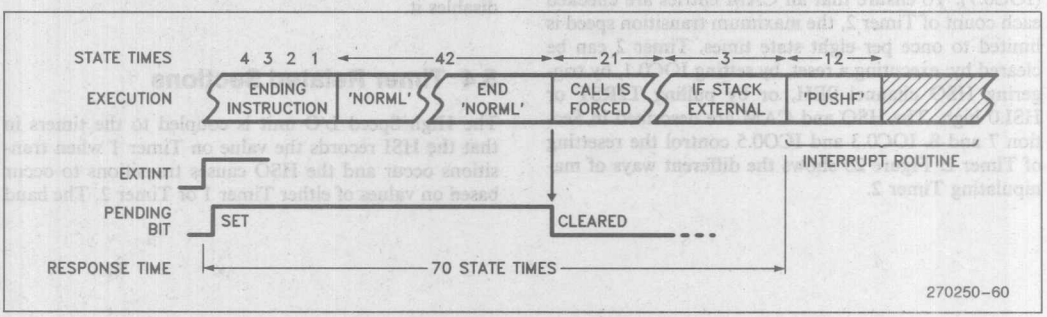


Figure 22A. Interrupt Response Time

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The "DI", "PUSHF", "POPF" and "TRAP" instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

5.0 TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

5.1 Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFXH and should not be used in programs.

5.2 Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising *and* falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in Section 7 and 8. IOC0.3 and ICO0.5 control the resetting of Timer 2. Figure 23 shows the different ways of manipulating Timer 2.

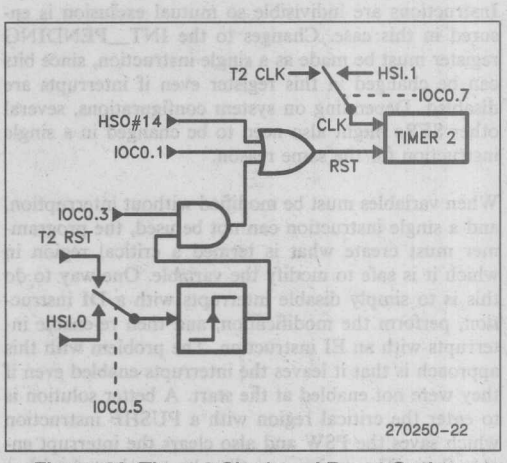


Figure 23. Timer 2 Clock and Reset Options

5.3 Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears bits 0 through 5 including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

5.4 Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The baud

rate generator can use the T2CLK pin as input to its counter. a complete listing of the functions of IOS1, IOC0, and IOC1 are in Section 11.

6.0 HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 are bidirectional pins which can also be used as HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 24.

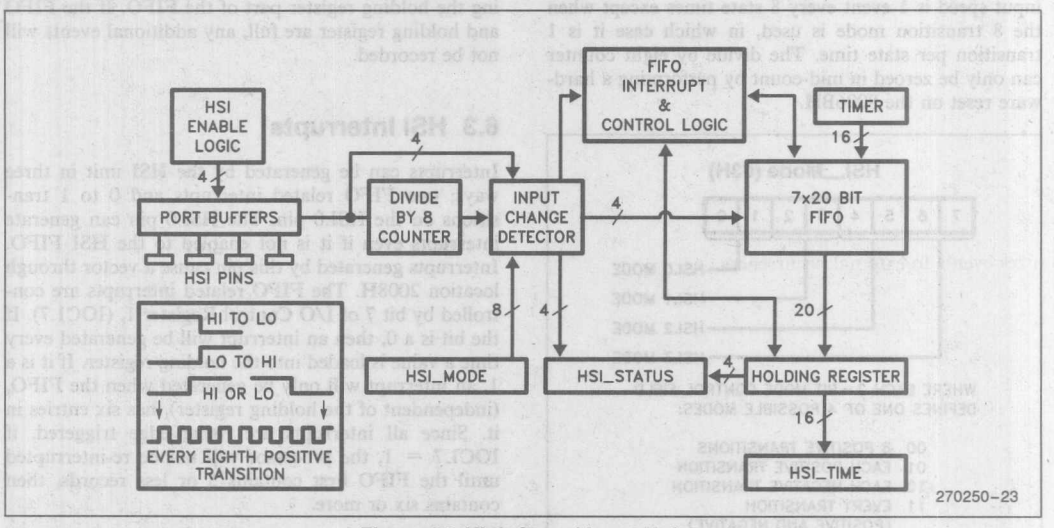


Figure 24. High Speed Input Unit

6.1 HSI Modes

There are 4 possible modes of operation for each of the HSI pins. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 25.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time. The divide by eight counter can only be zeroed in mid-count by performing a hardware reset on the 8096BH.

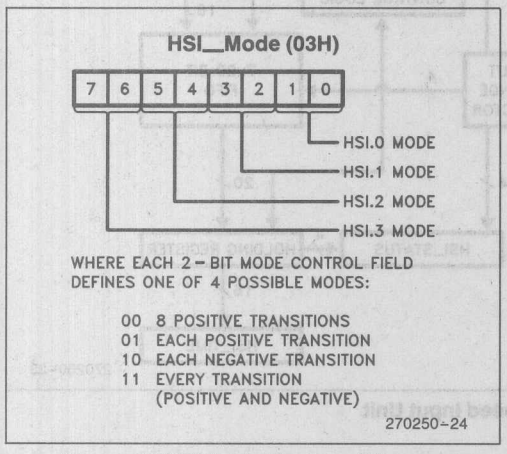


Figure 25. HSI Mode Register Diagram

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 26 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.

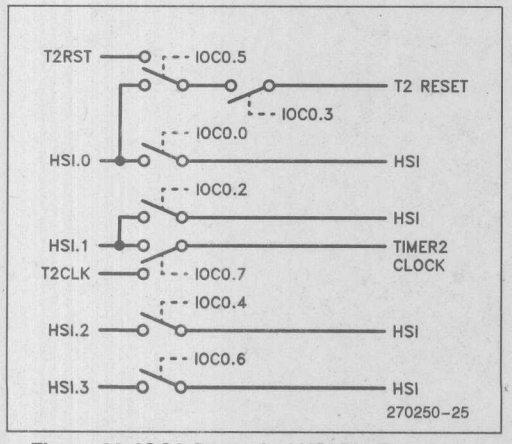


Figure 26. IOC0 Control of HSI Pin Functions

6.2 HSI FIFO

When an HSI event occurs, a 7×20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating which pins had events. It can take up to 8 state times for this information to reach the holding register. For this reason, 8 state times must be allowed between consecutive reads of HSI_TIME. When the FIFO is full, one additional event, for a total of 8 events, can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full, any additional events will not be recorded.

6.3 HSI Interrupts

Interrupts can be generated by the HSI unit in three ways; two FIFO related interrupts and 0 to 1 transitions on the HSI.0 pin. The HSI.0 pin can generate interrupts even if it is not enabled to the HSI FIFO. Interrupts generated by this pin cause a vector through location 2008H. The FIFO related interrupts are controlled by bit 7 of I/O Control Register 1, (IOC1.7). If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more.

6.4 HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the HSI holding register has data available to be read. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears bits 0–5, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 11.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI_STATUS Register is shown in Figure 27. Second, the HSI Time register is read. Reading the Time register unloads one level of the FIFO, so if the Time register is read before the Status register, the event information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI_TIME register is read without the holding register being loaded, the returned value will be indeterminate. Under the same conditions, the four bits in

HSI_STATUS indicating which events have occurred will also be indeterminate. The four HSI_STATUS bits which indicate the current state of the pins will always return the correct value.

It should be noted that many of the Status register conditions are changed by a reset, see Section 13. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11.

7.0 HIGH SPEED OUTPUTS

The High Speed Output unit, (HSO), is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, and switching 6 output lines (HSO.0 through HSO.5). Up to eight events can be pending at one time and interrupts can be generated whenever any of these events are triggered. HSO.4 and HSO.5 are bidirectional pins which can also be used as HSI.2 and HSI.3 respectively. Bits 4 and 6 of I/O Control Register 1, (IOC1.4, IOC1.6), enable HSO.4 and HSO.5 as outputs.

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate one or more of the six output pins. The other HSO interrupt is the software timer interrupt (vector = (200BH)) which is generated (if enabled) by any other HSO command, (e.g. triggering the A/D, resetting Timer 2 or generating a software time delay).

7.1 HSO CAM

A block diagram of the HSO unit is shown in Figure 28. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with the timer values every state time, taking 8 state times to compare all CAM registers with the timers. This defines the time resolution of the HSO to be 8 state times (2.0 microseconds at an oscillator frequency of 12 MHz).

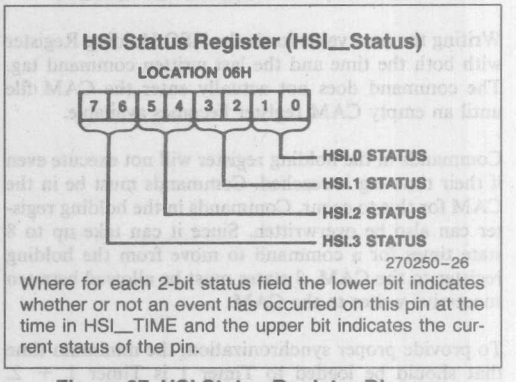


Figure 27. HSI Status Register Diagram

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the

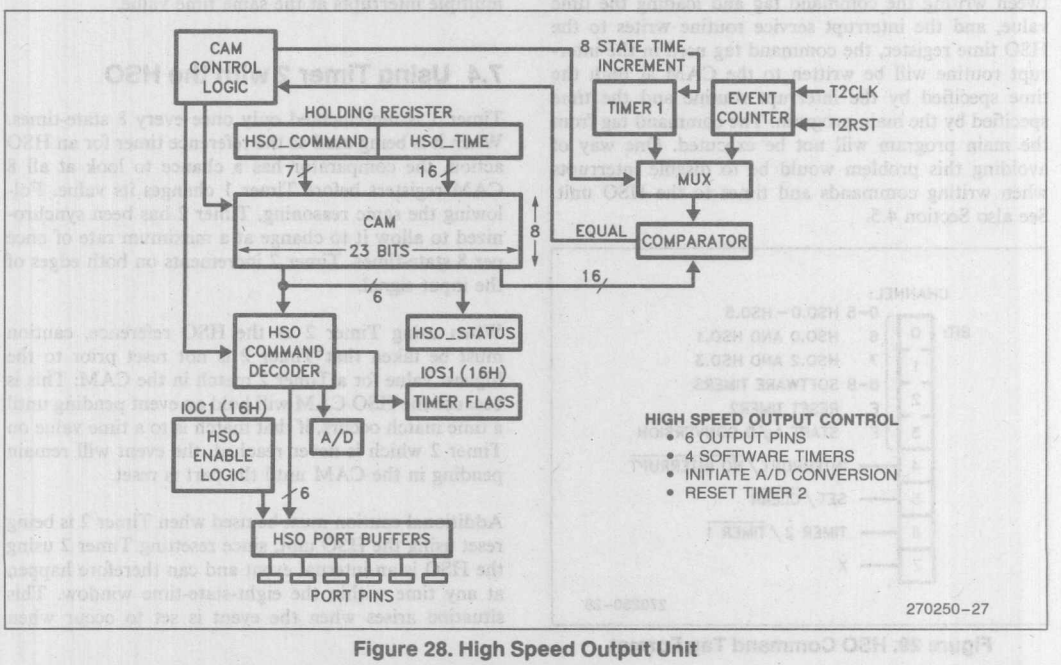


Figure 28. High Speed Output Unit

command to the HSO unit is shown in Figure 29. Note that bit 5 is ignored for command channels 8 through 0FH.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. The typical code would be:

```
LDB HSO_COMMAND,#what_to_do
ADD HSO_TIME,TIMER1,#when_to_do_it
```

Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Commands in the holding register can also be overwritten. Since it can take up to 8 state times for a command to move from the holding register to the CAM, 8 states must be allowed between successive writes to the CAM.

To provide proper synchronization, the minimum time that should be loaded to Timer 1 is Timer 1 + 2. Smaller values may cause the Timer match to occur 65,636 counts later than expected. A similar restriction applies if Timer 2 is used.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 4.5.

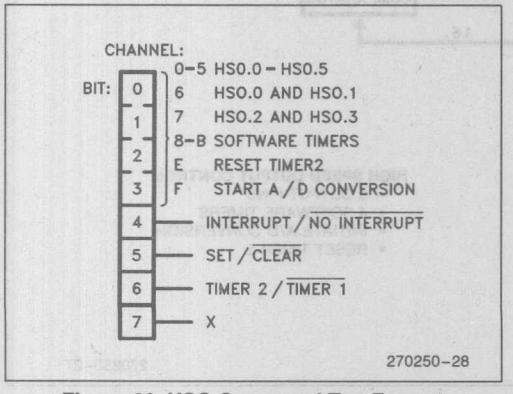


Figure 29. HSO Command Tag Format

7.2 HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in Section 11. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

The programmer should carefully decide which of these two flags is the best to use for each application.

7.3 Clearing the HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset. If, as an example, a command has been issued to set HSO.1 when TIMER 1 = 1234, then entering a second command which clears HSO.1 when TIMER 1 = 1234 will result in no operation on HSO.1. Both commands will remain in the CAM until TIMER 1 = 1234.

Internal events are not synchronized to Timer 1, and therefore cannot be cleared. This includes events on HSO channels 8 through F and all interrupts. Since interrupts are not synchronized it is possible to have multiple interrupts at the same time value.

7.4 Using Timer 2 with the HSO

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

When using Timer 2 as the HSO reference, caution must be taken that Timer 2 is not reset prior to the highest value for a Timer 2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. This situation arises when the event is set to occur when

Timer 2 is equal to zero. If HSI.0 or the T2RST pin is used to clear Timer 2, and Timer 2 equal to zero triggers the event, then the event may not occur. This is because HSI.0 and T2RST clear Timer 2 asynchronously, and Timer 2 may then be incremented to one before the HSO CAM entry can be read and acted upon. This can be avoided by setting the event to occur when Timer 2 is equal to one. This method will ensure that there is enough time for the CAM entry recognition.

The same asynchronous nature can affect events scheduled to occur at the same time as an internal Timer 2 reset. These events should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from "N" to zero, and then changed to a one on the next transition.

7.5 Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear bits 0 through 5. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 11.5.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11. The Timers are described in Section 5 and the HSI is described in Section 6.

8.0 ANALOG INTERFACE

The 8096H can easily interface to analog signals using its Analog to Digital Converter and its Pulse-Width-Modulated (PWM) output and HSO Unit. Analog inputs are accepted by the 8-input, 10-bit A to D converter. The PWM and HSO units provide digital signals which can be filtered for use as analog outputs.

8.1 Analog Inputs

A to D conversion is performed on one of the 8 inputs at a time using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on selected members of the MCS-96 family. See Section 14 for the device selection matrix.

Each conversion on the 8096BH requires 88 state-times (22 μs at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held nominally at 5V. The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage-ANGND})/(\text{VREF-ANGND})$$

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

ANGND must be tied to VSS (digital ground) in order for the 8096BH to operate properly. This common connection should be made as close to the chip as possible, and using good bulk and high frequency by-pass capacitors to decouple power supply variations and noise from the circuit. Analog design rules call for one and only one common connection between analog and digital returns to eliminate unwanted ground variations.

The A/D converter has sample and hold. The sampling window is open for 4 state times which are included in the 88 state-time conversion period. The exact timings of the A/D converter can be found in Section 3 of the Hardware Design chapter.

8.2 A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see Sections 4 and 11). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #0FH) triggers it. The A/D command register must be written to for each conversion, even if the HSO is used as the trigger. A to D commands are formatted as shown in Figure 30.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

8.3 A/D Results

Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 31. Note that the status bit may not be set until 8 state

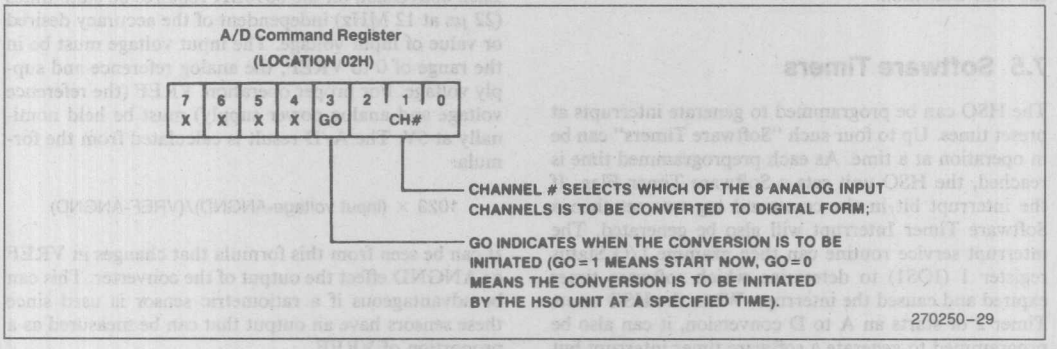


Figure 30. A/D Command Register

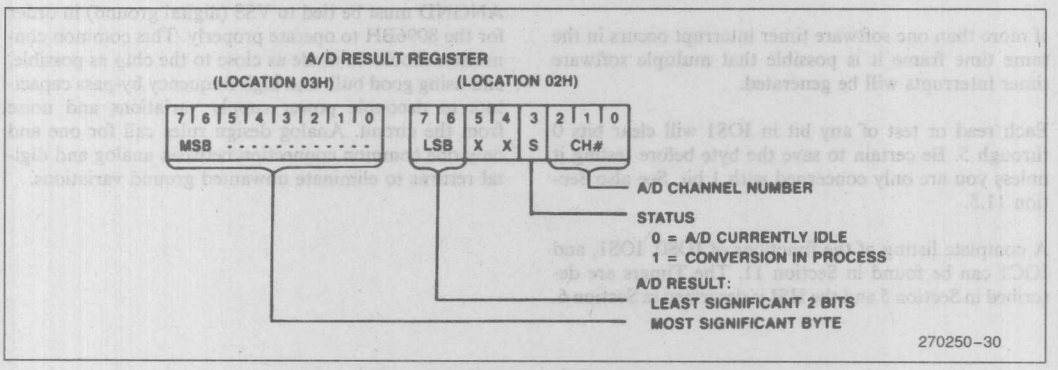


Figure 31. A/D Result Register

times after the go command, so it is necessary to wait 8 state times before testing it. Information on using the HSO is in Section 7.

8.4 Pulse Width Modulation Output (D/A)

Digital to analog conversion can be done with the Pulse Width Modulation output; a block diagram of the circuit is shown in Figure 32. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 33. Note that when the PWM register equals 00, the output is always low. Additionally, the PWM register will only be reloaded from the temporary latch when the counter overflows. This means that the compare circuit will not recognize a new value to compare against until the counter has expired the remainder of the current 8-bit count.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μs at 12 MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

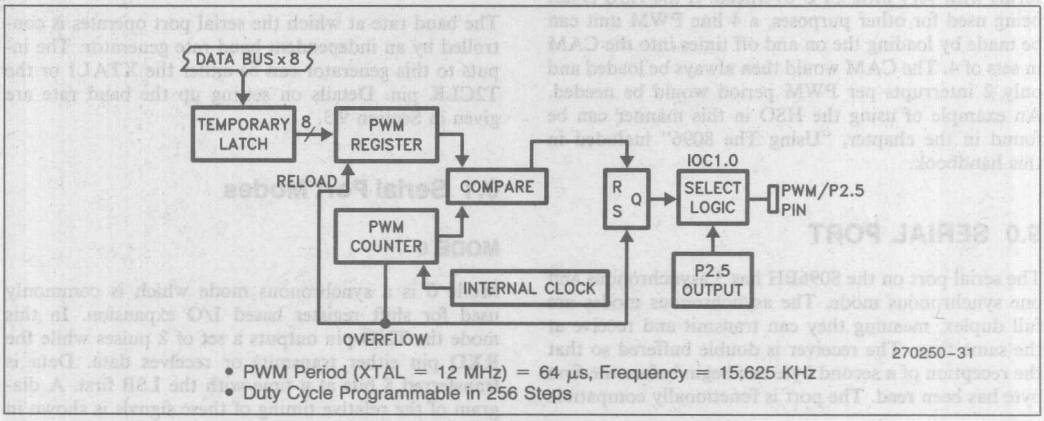


Figure 32. Pulse Width Modulated (D/A) Output

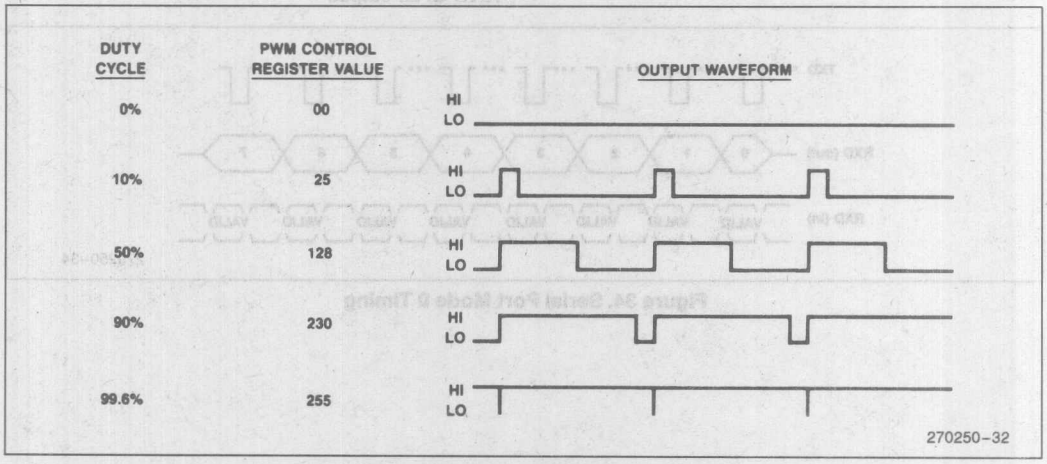


Figure 33. Typical PWM Outputs

Details about the hardware required for smooth, accurate D/A conversion can be found in Section 4 of the Hardware Design chapter. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in Section 11.

8.5 PWM Using the HSO

The HSO unit can be used to generate PWM waveforms with very little CPU overhead. If the HSO is not being used for other purposes, a 4 line PWM unit can be made by loading the on and off times into the CAM in sets of 4. The CAM would then always be loaded and only 2 interrupts per PWM period would be needed. An example of using the HSO in this manner can be found in the chapter, "Using The 8096" included in this handbook.

9.0 SERIAL PORT

The serial port on the 8096BH has 3 asynchronous and one synchronous mode. The asynchronous modes are full duplex, meaning they can transmit and receive at the same time. The receiver is double buffered so that the reception of a second byte can begin before the first byte has been read. The port is functionally compatible with the serial port on the MCS-51 family of microcontrollers, although the software used to control the ports is different.

Control of the serial port is handled through the Serial Port Control/Status Register at location 11H. Figure 37 shows the layout of this register. The details of using it to control the serial port will be discussed in Section 9.2.

Data to and from the serial port is transferred through SBUF (rx) and SBUF (tx), both located at 07H. Although these registers share the same address, they are physically separate, with SBUF (rx) containing the data received by the serial port and SBUF (tx) used to hold data ready for transmission. The program cannot write to SBUF (rx) or read from SBUF (tx).

The baud rate at which the serial port operates is controlled by an independent baud rate generator. The inputs to this generator can be either the XTAL1 or the T2CLK pin. Details on setting up the baud rate are given in Section 9.3.

9.1 Serial Port Modes

MODE 0

Mode 0 is a synchronous mode which is commonly used for shift register based I/O expansion. In this mode the TXD pin outputs a set of 8 pulses while the RXD pin either transmits or receives data. Data is transferred 8 bits at a time with the LSB first. A diagram of the relative timing of these signals is shown in Figure 34. Note that this is the only mode which uses RXD as an output.

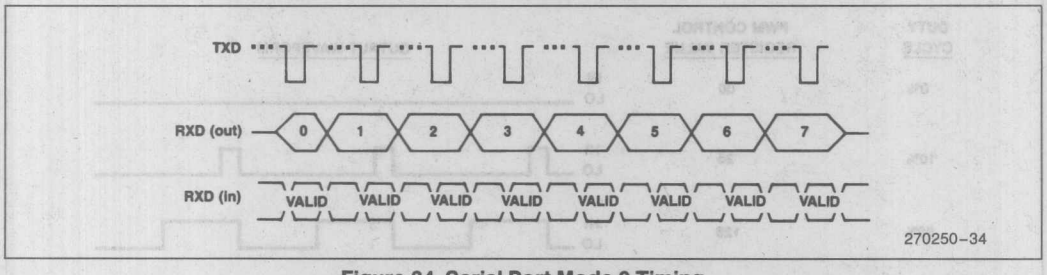


Figure 34. Serial Port Mode 0 Timing

Although it is not possible to transmit and receive at the same time using this mode, two external gates and a port pin can be used to time-multiplex the two functions. An example of multiplexing transmit and receive is discussed in Section 6.1 of the Hardware Design chapter.

MODE 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode is shown in Figure 35. It consists of 10 bits; a start bit (0), 8 data bits (LSB first), and a stop bit (1). If parity is enabled, (the PEN bit is set to a 1), an even parity bit is sent instead of the 8th data bit and parity is checked on reception.

MODE 2

Mode 2 is the asynchronous 9th bit recognition mode. This mode is commonly used with Mode 3 for multiprocessor communications. Figure 36 shows the data frame used in this mode. It consists of a start bit (0), 9 data bits (LSB first), and a stop bit (1). When transmitting, the 9th bit can be set to a one by setting the TB8 bit in the control register before writing to SBUF (tx). The TB8 bit is cleared on every transmission, so it must be set prior to writing to SBUF (tx) each time it is desired. During reception, the serial port interrupt and the Receive Interrupt (RI) bit will not be set unless the 9th bit being received is set. This provides an easy way to have selective reception on a data link. Parity cannot be enabled in this mode.

MODE 3

Mode 3 is the asynchronous 9th bit mode. The data frame for this mode is identical to that of Mode 2. The transmission differences between Mode 3 and Mode 2 are that parity can be enabled (PEN = 1) and cause the 9th data bit to take the even parity value. The TB8 bit can still be used if parity is not enabled (PEN = 0). When in Mode 3, a reception always causes an interrupt, regardless of the state of the 9th bit. The 9th bit is stored if PEN = 0 and can be read in bit RB8. If PEN = 1 then RB8 becomes the Receive Parity Error (RPE) flag.

9.2 Controlling the Serial Port

Control of the serial port is done through the Serial Port Control (SP_CON) and Serial Port Status (SP_STAT) registers shown in Figure 37. Writing to location 11H accesses SP_CON while reading it access SP_STAT. Note that reads of SP_STAT will return indeterminate data in the lower 5 bits and writing to the upper 3 bits of SP_CON has no effect on chip functionality. The TB8 bit is cleared after each transmission and both TI and RI are cleared whenever SP_STAT (not SP_CON) is accessed. Whenever the TXD pin is used for the serial port it must be enabled by setting IOC1.5 to a 1. IOC1 is discussed further in Section 11.3. Information on the hardware connections and timing of the serial port is in Section 6 of the Hardware Design chapter.

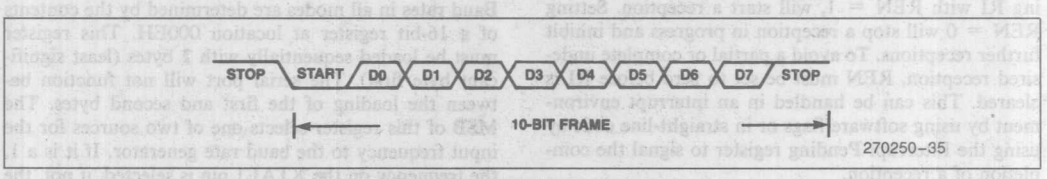


Figure 35. Serial Port Frame—Mode 1

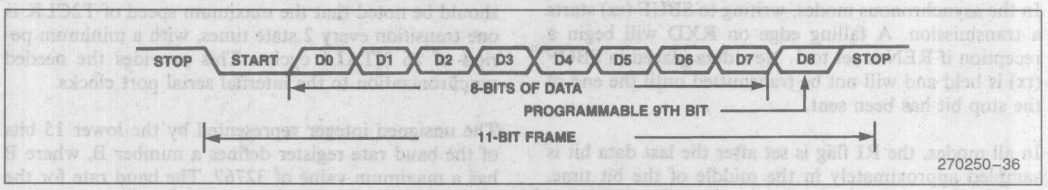
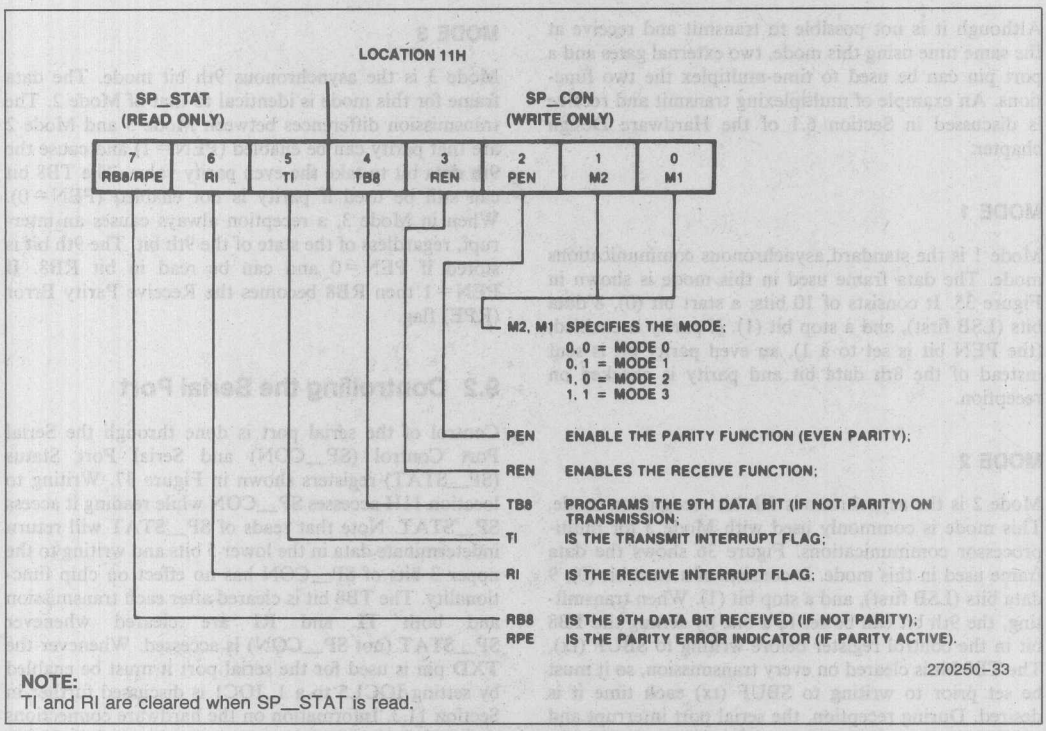


Figure 36. Serial Port Frame Modes 2 and 3



NOTE:
TI and RI are cleared when SP_STAT is read.

Figure 37. Serial Port Control/Status Register

In Mode 0, if REN = 0, writing to SBUF (tx) will start a transmission. Causing a rising edge on REN, or clearing RI with REN = 1, will start a reception. Setting REN = 0 will stop a reception in progress and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before RI is cleared. This can be handled in an interrupt environment by using software flags or in straight-line code by using the Interrupt Pending register to signal the completion of a reception.

In the asynchronous modes, writing to SBUF (tx) starts a transmission. A falling edge on RXD will begin a reception if REN is set to 1. New data placed in SBUF (tx) is held and will not be transmitted until the end of the stop bit has been sent.

In all modes, the RI flag is set after the last data bit is sampled approximately in the middle of the bit time. Also for all modes, the TI flag is set after the last data bit (either 8th or 9th) is sent, also in the middle of the bit time. The flags clear when SP_STAT is read, but do not have to be clear for the port to receive or transmit. The serial port interrupt bit is set as a logical OR of the RI and TI bits. Note that changing modes will reset the Serial Port and abort any transmission or reception in progress on the channel. If the T_X and R_X pins are tied together for loopback testing, the RI flag will be written first.

9.3 Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The serial port will not function between the loading of the first and second bytes. The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles. This provides the needed synchronization to the internal serial port clocks.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4 * (B + 1)}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64 * (B + 1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16 * B}; \quad B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12 MHz, are shown below.

Baud Rate	Baud Register Value	
	Mode 0	Others
9600	8137H	8013H
4800	8270H	8026H
2400	84E1H	804DH
1200	89C3H	809BH
300	A70FH	8270H

The maximum baud rates are 1.5 Mbaud synchronous and 187.5 Kbaud asynchronous with 12 MHz on XTAL1.

9.4 Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In Mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. Slaves in Mode 2 will not be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to Mode 3 to receive the coming data frames, while the slaves that were not addressed stay in Mode 2 and go on about their business.

10.0 I/O PORTS

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some are output only, some are bidirectional and some have alternate functions. In addition to these ports, the HSI/O unit can be used to provide extra I/O lines if the timer related features of these lines are not needed.

Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that hold the bits to be output. Bidirectional ports consist of an internal register, an input buffer, and an output buffer.

Port 0 is an input port which is also used as the analog input for the A to D converter. Port 1 is a quasi-bidirectional port. Port 2 contains three types of port lines: quasi-bidirectional, input and output. The input and output lines are shared with other functions in the 8096BH as shown in Table 4. Ports 3 and 4 are open-drain bidirectional ports which share their pins with the address/data bus.

Table 4. Port 2 Alternate Functions

Port	Function	Alternate Function	Controlled by
P2.0	Output	TXD (Serial Port Transmit)	IOC1.5
P2.1	Input	RXD (Serial Port Receive)	N/A
P2.2	Input	EXTINT (External Interrupt)	IOC1.1
P2.3	Input	T2CLK (Timer 2 Input)	IOC0.7
P2.4	Input	T2RST (Timer 2 Reset)	IOC0.5
P2.5	Output	PWM (Pulse-Width Modulation)	IOC1.0
P2.6	Quasi-Bidirectional		
P2.7	Quasi-Bidirectional		

Section 2 of the Hardware Design chapter contains additional information on the timing, drive capabilities, and input impedances of I/O pins.

10.1 Input Ports

Input ports and pins can only be read. There are no output drivers on these pins. The input leakage of these pins is in the microamp range. The specific values can be found in the data sheet for the device being considered.

In addition to acting as a digital input, each line of Port 0 can be selected to be the input of the A to D converter as discussed in Section 8. The pins on Port 0 are tested

to have D.C. leakage of 3 microamps or less, as specified in the data sheet for the device being considered. The capacitance on these pins is approximately 5 pF and will instantaneously increase by around 5 pF when the pin is being sampled by the A to D converter.

The 8096BH samples the input to the A/D for 4 state times at the beginning of the conversion. Details on the A to D converter can be found in Section 8 of this chapter and in Section 3 of the Hardware Design chapter.

10.2 Quasi-Bidirectional Ports

Port 1, Port 2.6 and Port 2.7 are quasi-bidirectional ports. "Quasi-bidirectional" means that the port pin has a weak internal pullup that is always active and an internal pulldown which can be on to output a 0, or off to output a 1. If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown. If the external pulldown is on, it will input a 0 to the 8096BH, if it is off, a 1 will be input. From the user's point of view, the main difference between a quasi-bidirectional port and a standard input port is that the quasi-bidirectional port will source current if externally pulled low. It will also pull itself high if left unconnected.

In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time. When this pullup is on the pin can typically source 30 milliamps to V_{SS}.

When the processor writes to the pins of a quasi-bidirectional port it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to its associated SFR bit, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pullup device which can be easily driven down by the device driving the input.

If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port.

Particular care should be exercised when using XOR opcodes or any opcode which is a read-modify-write instruction. It is possible for a Quasi-Bidirectional Pin to be written as a one, but read back as a zero if an external device (i.e., a transistor base) is pulling the pin below V_{IH}. See the Hardware Design Chapter Section 2.2 for further details on using the Quasi-Bidirectional Ports.

10.3 Output Ports

Output pins include the bus control lines, the HSO lines, and some of Port 2. These pins can only be used as outputs as there are no input buffers connected to them. It is not possible to use immediate logical instructions such as XOR PORT2, #00111B to toggle these pins. The output currents on these ports is higher than that of the quasi-bidirectional ports.

10.4 Ports 3 and 4/AD0–15

These pins have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accesing off-chip memory. If the $\overline{EA}$ line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Accessing Port 3 and 4 as I/O is easily done from internal registers. Since the LD and ST instructions require the use of internal registers, it may be necessary to first move the port information into an internal location before utilizing the data. If the data is already internal, the LD is unnecessary. For instance, to write a word value to Port 3 and 4 . . .

```
LD intreg, portdata   ; register ← data
                      ; not needed if already internal

ST intreg, 1FFEH      ; register → Port 3 and 4
```

To read Port 3 and 4 requires that "ones" be written to the port registers to first setup the input port configuration circuit. Note that the ports are reset to this input condition, but if zeroes have been written to the port, then ones must be re-written to any pins which are to be used as inputs. Reading Port 3 and 4 from a previously written zero condition is as follows . . .

```
LD intregA, #0FFFFH   ; setup port change mode pattern

ST intregA, 1FFEH     ; register → Port 3 and 4
                      ; LD & ST not needed if previously
                      ; written as ones

LD intregB, 1FFEH     ; register ← Port 3 and 4
```

Note that while the format of the LD and ST instructions are similar, the source and destination directions change.

When acting as the system bus the pins have strong drivers to both V_{CC} and V_{SS}. These drivers are used whenever data is being output on the system bus and are not used when data is being output by Ports 3 and 4. Only the pins and input buffers are shared between the bus and the ports. The ports use different output buffers which are configured as open-drain, and require pullup resistors. (open-drain is the MOS version of open-collector.) The port pins and their system bus functions are shown in Table 5.

Table 5. P3,4/AD0–15 Pins

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

11.0 STATUS AND CONTROL REGISTERS

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

11.1 I/O Control Register 0 (IOC0)

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 38.

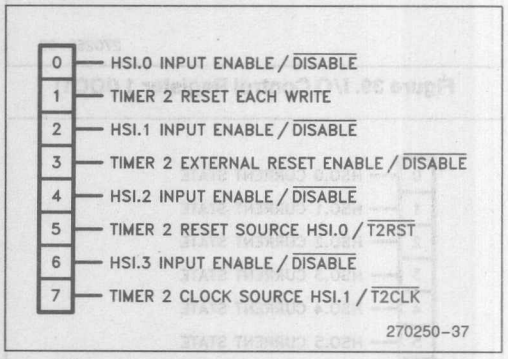


Figure 38. I/O Control Register 0 (IOC0)

11.2 I/O Control Register 1 (IOC1)

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in Section 4. The positions of the IOC1 control bits are shown in Figure 39.

11.3 I/O Status Register 0 (IOS0)

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 40.

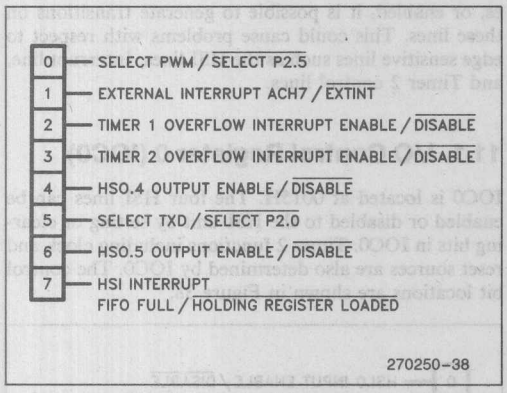


Figure 39. I/O Control Register 1 (IOC1)

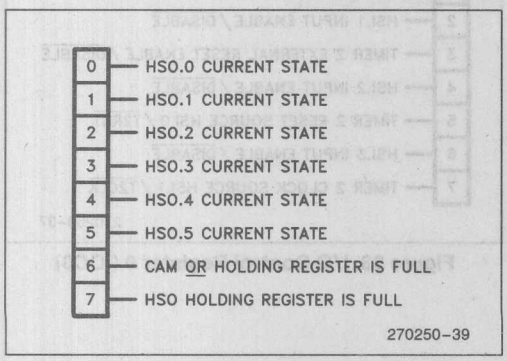


Figure 40. I/O Status Register 0 (IOS0)

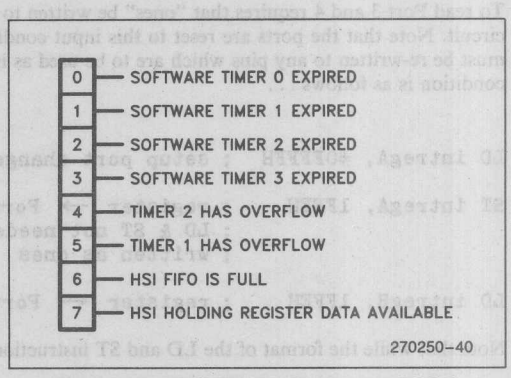


Figure 41. HSIO Status Register 1 (IOS1)

11.4 I/O Status Register 1 (IOS1)

IOS1 is located at 016H. It contains status bits for the timers and the HSI/O. The positions of these bits are shown in Figure 41.

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LDB    AL,IOS1
```

but also to implicit reads such as:

```
JB     IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB    IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on, then the modification of IOS1_image must be done from inside a critical region (see Section 4.4).

12.0 WATCHDOG TIMER

The WatchDog Timer (WDT) provides a means to recover gracefully from a software upset. When the watchdog is enabled it will initiate a hardware reset unless the software clears it every 64K state times.

The WDT is implemented as an 8-bit timer with an 8-bit prescaler. The prescaler is not synchronized, so the timer will overflow between 65280 and 65535 state times after being reset. When the timer overflows it pulls down the $\overline{\text{RESET}}$ pin for at least two state times, resetting the 8096BH and any other devices tied to the $\overline{\text{RESET}}$ line. If a large capacitor is connected to the line, the pin may take a long time to go low. This will effect the length of time the pin is low and the voltage on the pin when it is finished falling. Section 1.4 of the Hardware Design chapter contains more information about reset hardware connections.

The WDT is enabled the first time it is cleared. Once it is enabled, it can only be disabled by resetting the 8096BH. The internal bit which controls the watchdog can typically maintain its state through power glitches as low as V_{SS} and as high as 7.0V for up to one millisecond.

Enabling and clearing the WDT is done by writing a "01EH" followed by a "0E1H" to the WDT register at location 0AH. This double write is used to help prevent accidental clearing of the timer.

12.1 Software Protection Hints

Glitches and noise on the PC board can cause software upsets, typically by changing either memory locations or the program counter. These changes can be internal to the chip or be caused by bad data returning to the chip.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 milliseconds if a 12 MHz oscillator is used.

When using the WDT to protect software it is desirable to reset it from only one place in code. This will lessen the chance that an undesired WDT reset will occur. The section of code that resets the WDT should monitor the other code sections for proper operation. This can be done by checking variables to make sure they are within reasonable values. Simply using a software timer to reset the WDT every 15 milliseconds will not provide much protection against minor problems.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed undesired results will occur. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. Since RST is a one-byte instruction, the NOPs are not needed if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery should the processor have a glitch in the program flow. Since RST instruction has an opcode of 0FFH, pulling the data lines high with resistors will cause an RST to be executed if unimplemented memory is addressed.

12.2 Disabling The Watchdog

The watchdog should be disabled by software not initializing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the $\overline{\text{RESET}}$ pin at 2.0V to 2.5V. Voltages over 2.5V on the pin could quickly damage the part. Even at 2.5V, using this technique for other than debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products. Section 1.6 of the Hardware Design chapter has more information on disabling the Watchdog Timer.

13.0 RESET

13.1 Reset Signal

As with all processors, the 8096BH must be reset each time the power is turned on. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the inter-

nal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{EA}$ pin selects the external execution only mode the CCB is read from external memory.

The 8096BH can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2 state times longer than required for V_{CC} and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire ORed together. Remember, the $\overline{RESET}$ pin itself can be a reset source when the RST instruction is executed or when the Watchdog Timer overflows. Details of hardware suggestions for reset can be found in Section 1.4 of the Hardware Design chapter.

13.2 Reset Status

The I/O lines and control lines of the 8096BH will be in their reset state within 2 state times after reset is low, with V_{CC} and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

Register	Reset Value
Port 1	XXXXXXXXB
Port 2	XX0XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Port Control	XXXX0XXXB
Serial Port Status	X00XXXXXB
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B
HSI FIFO	empty
HSO CAM	empty
HSO SFR	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

Figure 42. Register Reset Status

Port 1 and Port 2.6, 2.7 reset to a floating or weak pull-up condition. HSO.4 and HSO.5 reset to a floating condition as they are disabled by IOC1.4 and IOC1.6.

Other conditions following a reset are:

Pin	Reset Value
$\overline{RD}$	high
$\overline{WR}/\overline{WRL}$	high
ALE/$\overline{ADV}$	high
$\overline{BHE}/\overline{WRH}$	low
INST	high
HSO Lines	XX0000B

Figure 43. Bus Control Pins Reset Status

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

13.3 Reset Sync Mode

The $\overline{RESET}$ line can be used to start the 8096BH at an exact state time to provide for synchronization of test equipment and multiple chip systems. $\overline{RESET}$ is active low. To synchronize parts, $\overline{RESET}$ is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in Section 1.5 of the Hardware Design chapter.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

14.0 QUICK REFERENCE

14.1 Pin Description

On the 48-pin parts the following pins are not bonded out: Port1, Port0 (Analog In) bits 0–3, T2CLK (P2.3), T2RST (P2.4), P2.6, P2.7, CLKOUT, INST, NMI, BUSWIDTH ($\overline{TEST}$ on 8X9X devices).

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). Two pins.
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{RESET}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{RESET}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized. See Section 2.3.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. See Section 8.
ANGND	Reference ground for the A/D converter. Should be held at nominally the same potential as V_{SS}. See Section 8.
V_{PP}	Programming voltage for the EPROM parts. It should be +12.75V when programming and will float to 5V otherwise. It should not be above 5.5V on other than EPROM parts. This pin must float in the application circuit.
XTAL1	Input of the oscillator inverter and of the internal clock generator. See Section 1.5.
XTAL2	Output of the oscillator inverter. See Section 1.5.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle. See Section 1.5
$\overline{RESET}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{RESET}$ has an internal pullup. See Section 13.
BUSWIDTH	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. If this pin is left unconnected, it will rise to V_{CC}. See Section 2.7.
NMI	A positive transition causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{EA}$ = +12.5V causes execution to begin in the Programming mode on EPROM parts. $\overline{EA}$ has an internal pulldown, so it goes to 0 unless driven otherwise.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses. See Section 2.7.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
$\overline{WR}/\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}/\overline{WRL}$ is activated only during external memory writes. See Section 2.7.
$\overline{BHE}/\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, BHE # = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. See Section 2.7.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the Memory Controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low. See Section 2.7.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming mode. See Section 6.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. See Section 7.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming mode. See Section 10.
Port 1	8-bit quasi-bidirectional I/O port. See Section 10.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode. See Section 10.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the programming mode. See Sections 2.7 and 10.

14.2 Pin List

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0–AD15, which are listed under Port 3 and Port 4.

Name	68-Pin PLCC	68-Pin PGA	48-Pin DIP
ACH0/P0.0	6	4	—
ACH1/P0.1	5	5	—
ACH2/P0.2	7	3	—
ACH3/P0.3	4	6	—
ACH4/P0.4/MOD.0	11	67	43
ACH5/P0.5/MOD.1	10	68	42
ACH6/P0.6/MOD.2	8	2	40
ACH7/P0.7/MOD.3	9	1	41
ALE/$\overline{ADV}$	62	16	34
ANGND	12	66	44
$\overline{BHE}$/$\overline{WRH}$	41	37	15
BUSWIDTH ($\overline{TEST}$)	64	14	—
CLKOUT	65	13	—
$\overline{EA}$	2	8	39
EXTINT/P2.2/$\overline{PROG}$	15	63	47
HSI.0	24	54	3
HSI.1	25	53	4
HSI.2/HSO.4	26	52	5
HSI.3/HSO.5	27	51	6
HSO.0	28	50	7
HSO.1	29	49	8
HSO.2	34	44	9
HSO.3	35	43	10
HSO.4/HSI.2	26	52	5
HSO.5/HSI.3	27	51	6
INST	63	15	—
NMI	3	7	—
PWM/P2.5/$\overline{PDO}$	39	39	13
PALE/P2.1/RXD	17	61	1
$\overline{PROG}$/P2.2/EXTNT	15	63	47
PVER/P2.0/TXD	18	60	2
P0.0/ACH0	6	4	—
P0.1/ACH1	5	5	—
P0.2/ACH2	7	3	—
P0.3/ACH3	4	6	—
P0.4/ACH4/MOD.0	11	67	43
PO.5/ACH5/MOD.1	10	68	42
PO.6/ACH6/MOD.2	8	2	40
PO.7/ACH7/MOD.3	9	1	41
P1.0	19	59	—
P1.1	20	58	—
P1.2	21	57	—
P1.3	22	56	—
P1.4	23	55	—
P1.5	30	48	—
P1.6	31	47	—
P1.7	32	46	—
P2.0/TXD/PVER	18	60	2
P2.1/RXD/PALE	17	61	1
P2.2/EXTINT	15	63	47
P2.3/T2CLK	44	34	—
P2.4/T2RST	42	36	—
P2.5/PWM/$\overline{PDO}$	39	39	13
P2.6	33	45	—
P2.7	38	40	—
P3.0/AD0 PVAL	60	18	32
P3.1/AD1 PVAL	59	19	31
P3.2/AD2 PVAL	58	20	30
P3.3/AD3 PVAL	57	21	29
P3.4/AD4 PVAL	56	22	28
P3.5/AD5 PVAL	55	23	27
P3.6/AD6 PVAL	54	24	26
P3.7/AD7 PVAL	53	25	25
P4.0/AD8 PVAL	52	26	24
P4.1/AD9 PVAL	51	27	23
P4.2/AD10 PVAL	50	28	22
P4.3/AD11 PVAL	49	29	21
P4.4/AD12 PVAL	48	30	20
P4.5/AD13 PVAL	47	31	19
P4.6/AD14 PVAL	46	32	18
P4.7/AD15 PVAL	45	33	17
$\overline{RD}$	61	17	33
READY	43	35	16
$\overline{RESET}$	16	62	48
RXD/P2.1	17	61	1
SALE/PVER/P2.0	18	60	2
$\overline{SPROG}$/$\overline{PDO}$/P2.5	39	39	13
TXD/P2.0/SALE	18	60	2
T2CLK/P2.3	44	34	—
T2RST/P2.4	42	36	—
V_{PP}	37	41	12
V_{CC}	1	9	38
V_{PD}	14	64	46
V_{REF}	13	65	45
V_{SS}	68	10	11
V_{SS}	36	42	37
$\overline{WR}$/$\overline{WRL}$	40	38	14
$\overline{WRH}$/$\overline{BHE}$	41	37	15
XTAL1	67	11	36
XTAL2	66	12	35

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, $\overline{TEST}$, T2CLK (P2.3), T2RST (P2.4).

14.3 Packaging

The MCS-96 products are available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown below. Section 14.4 shows the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

The MCS®-96 Family Nomenclature

		Without A/D	With A/D
ROMless 809XBH	48 Pin		C8095BH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM 839XBH	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM 879XBH	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC

Transistor Count

Device Type	# MOS Gates
839XBH/879XBH	120,000
809XBH	50,000

MTBF Calculations*

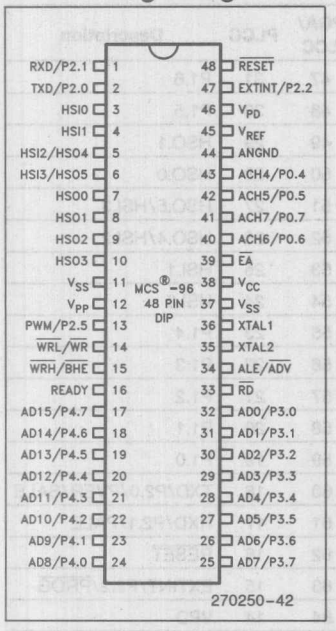
3.8×10^7 Device Hours @ 55°C
1.7×10^7 Device Hours @ 70°C

*MTBF data was obtained through calculations based upon the actual average junction temperatures under stress at 55°C and 70°C ambient.

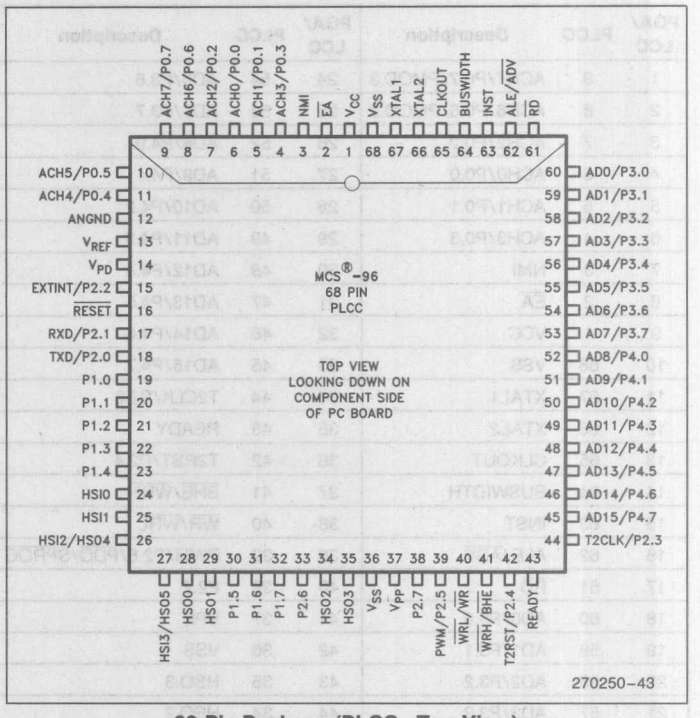
Thermal Characteristics

T_{CASE}		Package Type	θJa	θJc
COMM'L	EXPRESS			
85°C	100°C	PGA	35°C/W	10°C/W
85°C	100°C	PLCC	37°C/W	10°C/W
		LCC	28°C/W	—
		Plastic DIP	38°C/W	—
79.75°C	94.75°C	Ceramic DIP	26°C/W	6.5°C/W

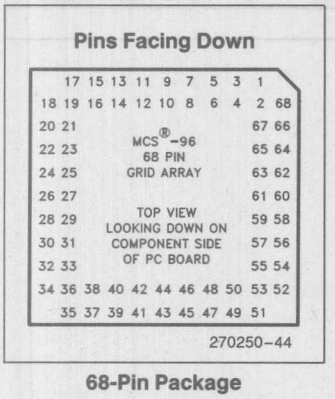
14.4 Package Diagrams



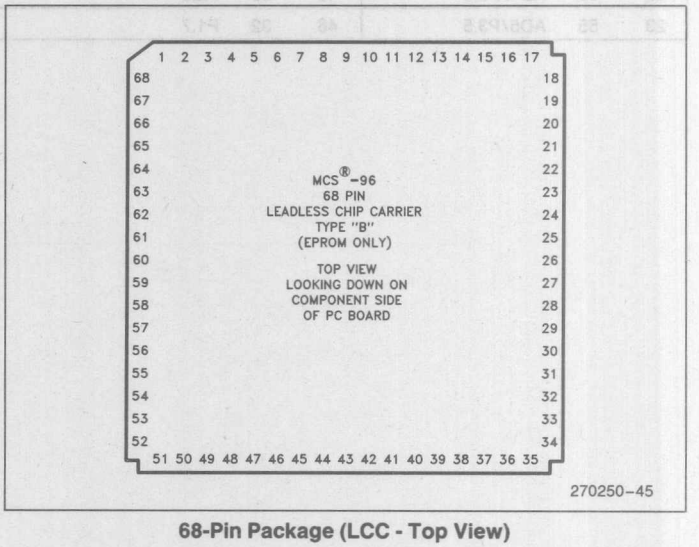
48-Pin Package



68-Pin Package (PLCC - Top View)



68-Pin Package
(Pin Grid Array - Top View)

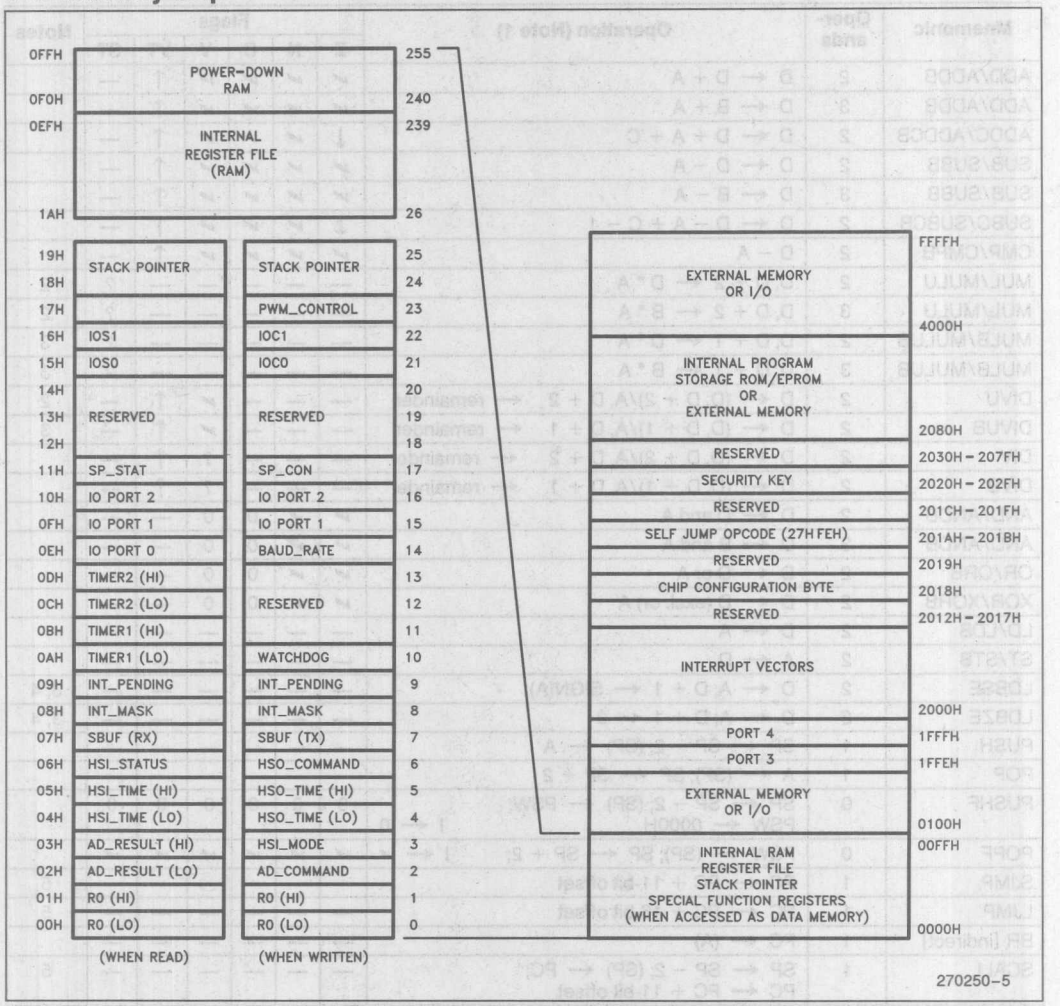


68-Pin Package (LCC - Top View)

14.5 PGA, PLCC and LCC Function Pinouts

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}/\overline{WRH}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{WR}/\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5/$\overline{PDO}$/$\overline{SPROG}$	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{PROG}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

14.6 Memory Map



14.7 Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↙	↙	↙	↙	↑	—	
ADD/ADDB	3	D ← B + A	↙	↙	↙	↙	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↙	↙	↙	↑	—	
SUB/SUBB	2	D ← D − A	↙	↙	↙	↙	↑	—	
SUB/SUBB	3	D ← B − A	↙	↙	↙	↙	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↙	↙	↙	↑	—	
CMP/CMPB	2	D − A	↙	↙	↙	↙	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↙	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↙	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↙	↙	0	0	—	—	
AND/ANDB	3	D ← B and A	↙	↙	0	0	—	—	
OR/ORB	2	D ← D or A	↙	↙	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↙	↙	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↙	↙	↙	↙	↙	↙	↙	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

14.8 Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270250-46

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

⑧ State times shown for 16-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑥					INDEXED⑥				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE①⑥ TIMES	BYTES	STATE①⑥ TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270250-47

NOTES:
① Number of state times shown for internal/external operands.
③ The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
⑤ State times for stack located internal/external.
⑥ State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.[8]

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.[8]

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	E0	3	5/9 STATE TIME (NOT TAKEN/TAKEN)[8]

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES[8]
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT[7]
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT[7]
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT[7]

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST[6]	FF	1	166	SKIP	00	2	4

NORMALIZE

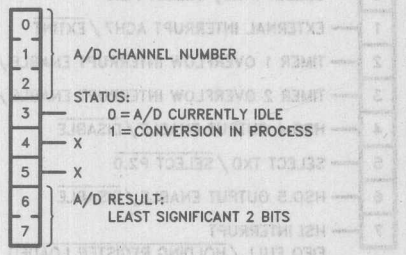
MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to $\overline{\text{RESET}}$, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

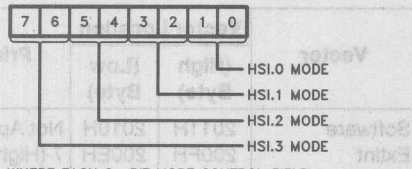
14.9 SFR Summary

A/D Result LO (02H)



270250-48

HSI_Mode (03H)

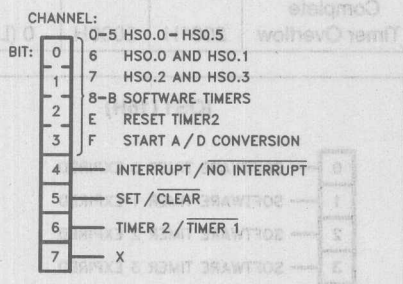


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

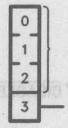
270250-49

HSO Command (06H)



270250-50

A/D Command (02H)

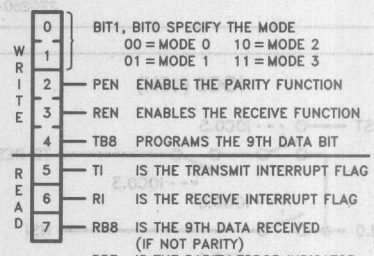


CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.

GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270250-51

SPCON/SPSTAT (11H)



270250-52

Baud Rate Calculations

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4*(B+1)};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

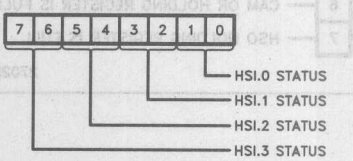
Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16*B};\ B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

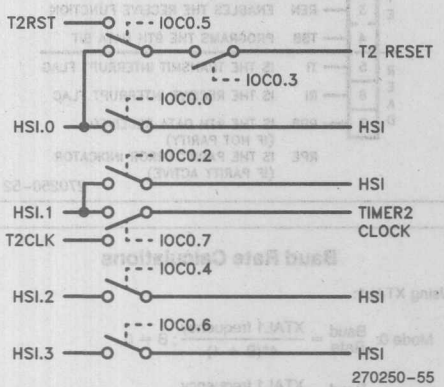
270250-53

IOC0 (15H)

0	HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1	TIMER 2 RESET EACH WRITE
2	HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4	HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5	TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6	HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7	TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$

270250-54

IOC0 (15H)



270250-55

IOS0 (15H)

0	HSO.0 CURRENT STATE
1	HSO.1 CURRENT STATE
2	HSO.2 CURRENT STATE
3	HSO.3 CURRENT STATE
4	HSO.4 CURRENT STATE
5	HSO.5 CURRENT STATE
6	CAM OR HOLDING REGISTER IS FULL
7	HSO HOLDING REGISTER IS FULL

270250-56

IOC1 (16H)

0	SELECT PWM / $\overline{\text{SELECT P2.5}}$
1	EXTERNAL INTERRUPT ACH7 / $\overline{\text{EXTINT}}$
2	TIMER 1 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
4	HSO.4 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
5	SELECT TXD / $\overline{\text{SELECT P2.0}}$
6	HSO.5 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
7	HSI INTERRUPT FIFO FULL / $\overline{\text{HOLDING REGISTER LOADED}}$

270250-57

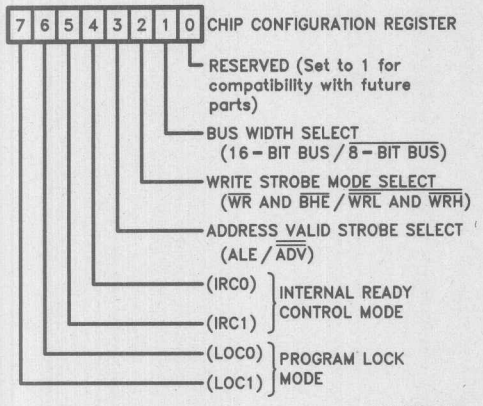
Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

IOS1 (16H)

0	SOFTWARE TIMER 0 EXPIRED
1	SOFTWARE TIMER 1 EXPIRED
2	SOFTWARE TIMER 2 EXPIRED
3	SOFTWARE TIMER 3 EXPIRED
4	TIMER 2 HAS OVERFLOW
5	TIMER 1 HAS OVERFLOW
6	HSI FIFO IS FULL
7	HSI HOLDING REGISTER DATA AVAILABLE

270250-58

Chip Configuration



Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Programming Function PMODE Values

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

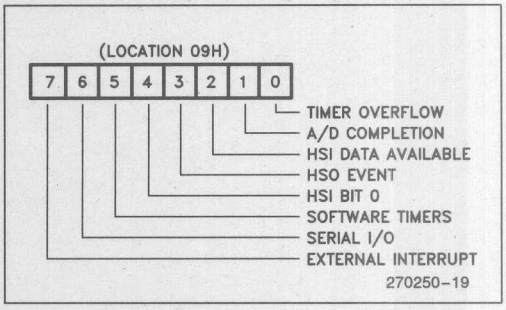
8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Port 2 Pin Functions

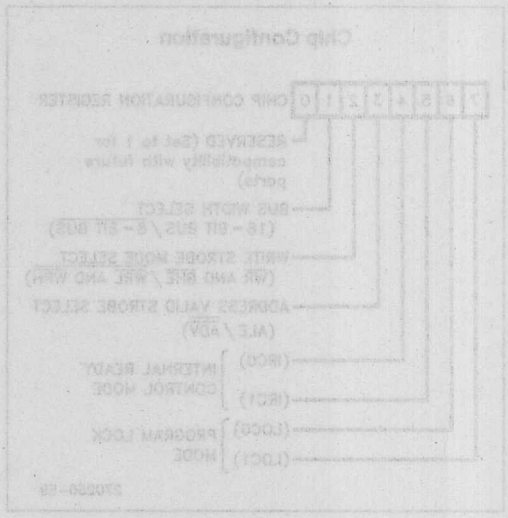
Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)

Interrupt Pending Register



PSW Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							


7
6
5
4
3
2
1
0
CHIP CONFIGURATION REGISTER
RESERVED (Set to 1 for compatibility with future parts)
BUS WIDTH SELECT (16-BIT BUS \ 8-BIT BUS)
WAIT STROBE MODE SELECT (WR AND BHE \ WR AND WRH)
ADDRESS VALID STROBE SELECT (ALE \ ADV)
INTERNAL READY CONTROL MODE (IRCO)
(IRC1)
PROGRAM LOCK MODE (LOCO)
(LOC1)

8096BH Hardware Design Information

2

2 80960BH Hardware Design Information

8096BH HARDWARE DESIGN INFORMATION

OVERVIEW

This Chapter of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this chapter a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

1.0 REQUIRED HARDWARE CONNECTIONS

Although the 8096BH is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 6 shows the connections that are needed for a single-chip system.

1.1 Power Supply Information

Power for the 8096BH flows through six pins; they are: three positive voltage pins—V_{CC} (digital), V_{REF} (Port 0 digital I/O and A/D power), V_{PD} (power down mode); and three common returns—two V_{SS} pins and one ANGND pin. All six of these pins **must** be connected on the 8096BH for normal operation. The V_{CC} pin, V_{REF} pin and V_{PD} pin should be tied to 5 volts. The two V_{SS} pins and the ANGND pin must be grounded. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The three common return pins should be connected at the chip with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between V_{SS1} and V_{SS2}. The two V_{SS} pins and the ANGND pin must all be nominally at 0 volts. The maximum current drain of the 8096BH is around 180 mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The V_{REF} pin supplies the digital circuitry in the A/D converter and provides the 5 volt reference to the analog portion of the converter. V_{REF} and ANGND must be connected even if the A/D converter is not used. More information on the analog power supply is in Section 3.1.

1.2 Other Needed Connections

Several other connections are needed to configure the 8096BH. In normal operation the following pins should be connected to the indicated power supply.

Pin	Power Supply
NMI	V_{CC}
$\overline{EA}$	V_{CC} (to allow internal execution) V_{SS} (to force external execution)

Although the $\overline{EA}$ pin has an internal pulldown, it is best to tie this pin to the desired level. This will prevent induced noise from disturbing the system. Raising $\overline{EA}$ to +12.75 volts will place an 8096BH in a special operating mode designed for programming and program memory verification (see Section 10).

1.3 Oscillator Information

The 8096BH requires a clock source to operate. This clock is provided to the chip through the XTAL1 input. The frequency of operation is from 6 MHz to 12 MHz.

The on-chip circuitry for the 8096BH oscillator is a single stage linear inverter as shown in Figure 1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 2. In this application, the crystal is being operated in its fundamental response mode as an inductive

reactance in parallel resonance with shunt capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 2) are not critical. Thirty pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

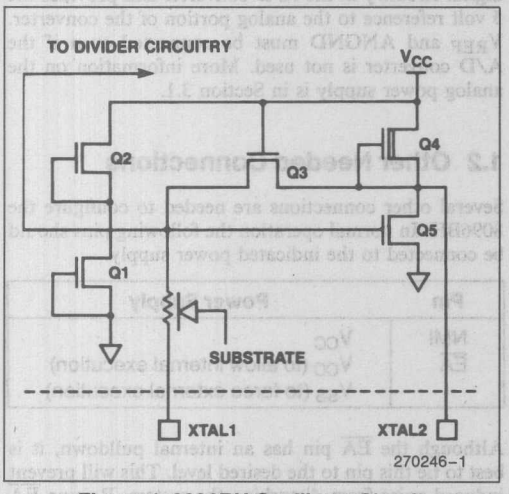


Figure 1. 8096BH Oscillator Circuit

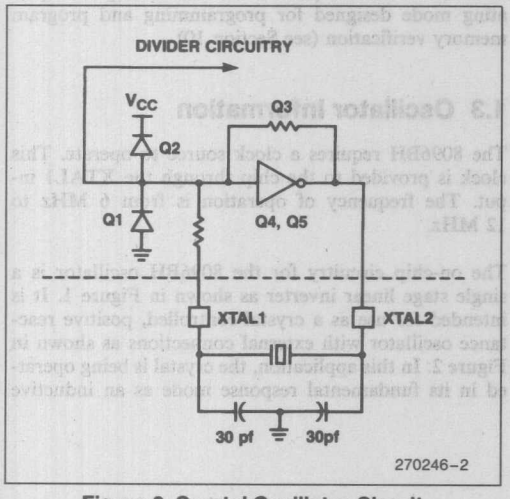


Figure 2. Crystal Oscillator Circuit

To drive the 8096BH with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels.

It is important that the minimum high and low times are met to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096BH internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4 shows the waveforms of the major internal timing signals.

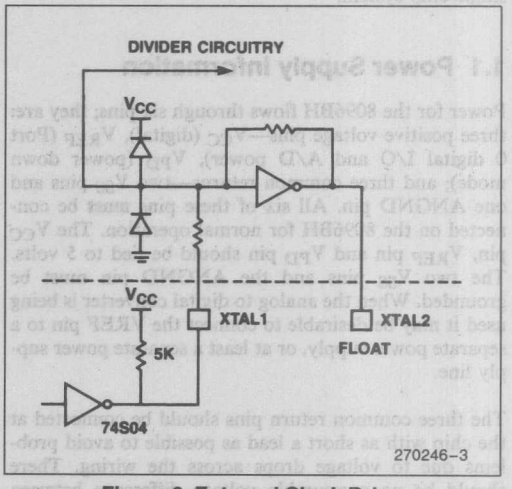


Figure 3. External Clock Drive

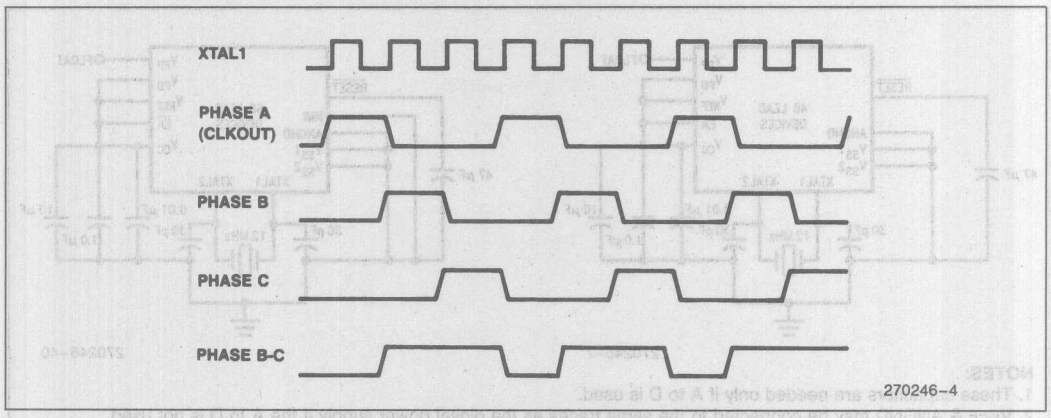


Figure 4. Internal Timings

1.4 Reset Information

In order for the 8096BH to function properly it must be reset. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the internal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{\text{EA}}$ pin selects the external execution only mode the CCB is read from external memory. See Figure 5.

There are several ways to provide a good reset to an 8096BH, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of V_{CC} is fast and the total reset time is less than around 50 milliseconds. It also may not work if the $\overline{\text{RESET}}$ pin is to be used to reset other parts on the board. An 8096BH with the minimum required connections is shown in Figure 6.

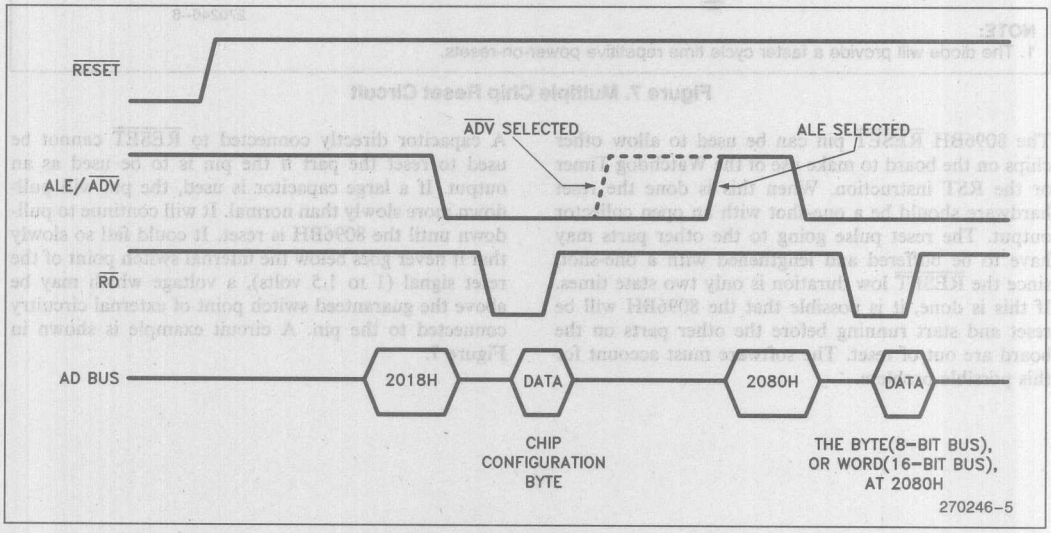
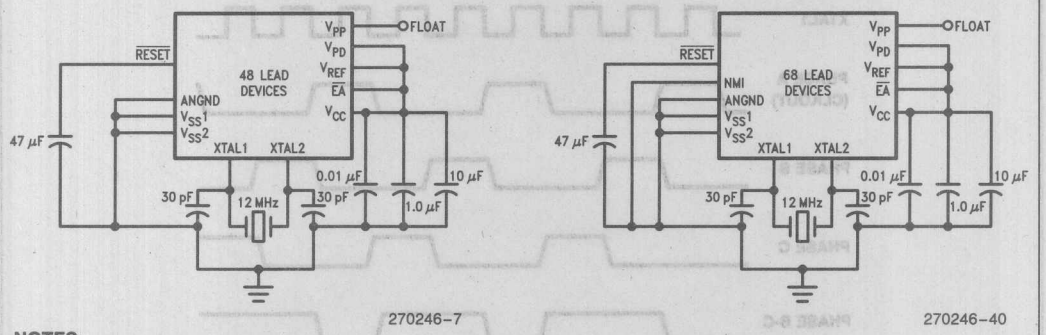
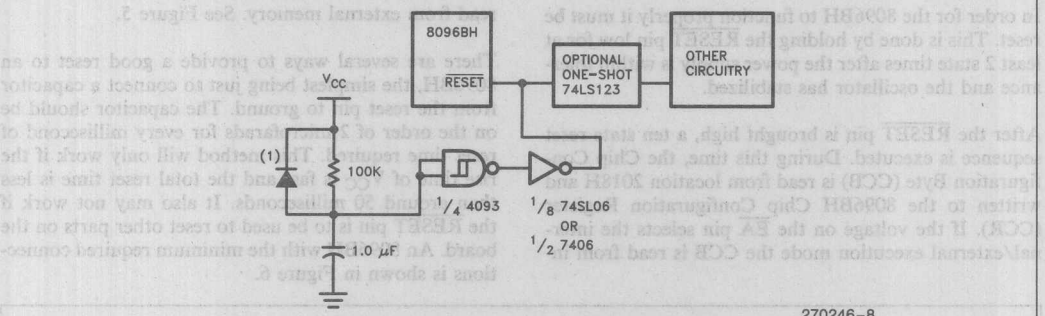


Figure 5. Reset Sequence



NOTES:
1. These capacitors are needed only if A to D is used.
2. V_{REF} & ANGND may be connected to the same traces as the digital power supply if the A to D is not used.

Figure 6. Minimum Hardware Connections



NOTE:
1. The diode will provide a faster cycle time repetitive power-on-resets.

Figure 7. Multiple Chip Reset Circuit

The 8096BH $\overline{\text{RESET}}$ pin can be used to allow other chips on the board to make use of the Watchdog Timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the $\overline{\text{RESET}}$ low duration is only two state times. If this is done, it is possible that the 8096BH will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to $\overline{\text{RESET}}$ cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull-down more slowly than normal. It will continue to pull-down until the 8096BH is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. A circuit example is shown in Figure 7.

1.5 Sync Mode

If $\overline{\text{RESET}}$ is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6½ XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 8. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

1.6 Disabling the Watchdog Timer

The Watchdog Timer will pull the $\overline{\text{RESET}}$ pin low when it overflows. See Figure 9. If the pin is being externally held above the low going threshold, the pulldown transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or $\overline{\text{RESET}}$ must be held high indefinitely. Just resetting the Watchdog Timer in software will not clear the flip-flop which keeps the $\overline{\text{RESET}}$ pulldown on.

The pulldown is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the Watchdog Timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pulldown is on, it should be through a resistor that will limit the voltage on $\overline{\text{RESET}}$ to 2.5 volts and the current through the pin to 40 milliamps.

If it is necessary to disable the Watchdog Timer for more than a brief test the software solution of never initiating the timer should be used. See Section 14 in the Architecture Chapter.

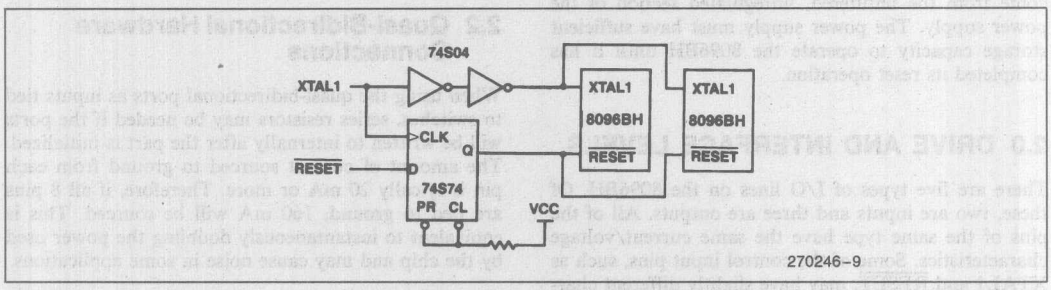


Figure 8. Reset Sync Mode

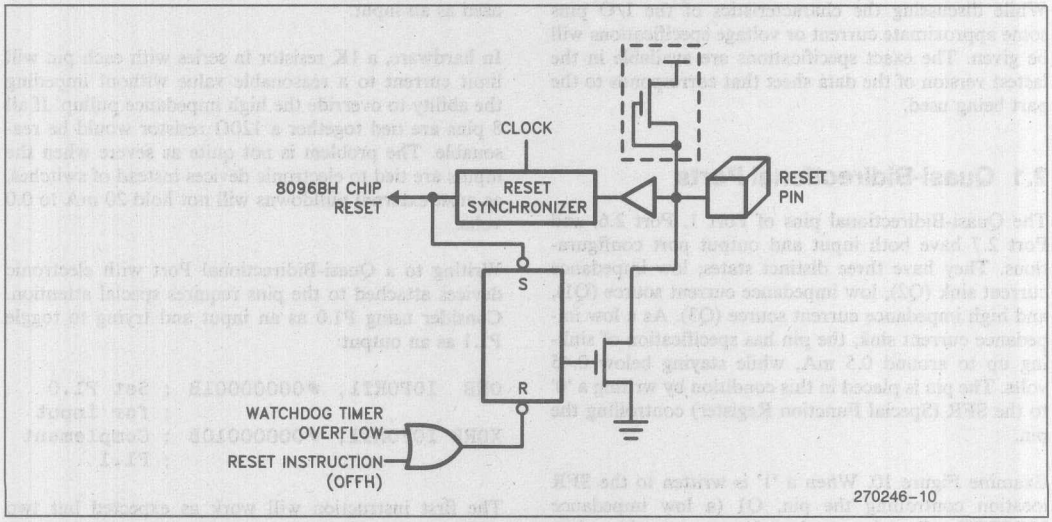


Figure 9. Reset Logic

1.7 Power Down Circuitry

Battery backup can be provided on the 8096BH with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the V_{PD} pin remains on. The required timings to put the part into power-down and an overview of this mode are given in Section 2.3 in the 8096BH Architecture Chapter.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

2.0 DRIVE AND INTERFACE LEVELS

There are five types of I/O lines on the 8096BH. Of these, two are inputs and three are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and $\overline{\text{RESET}}$, may have slightly different characteristics. These pins are discussed in Section 1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the lastest version of the data sheet that corresponds to the part being used.

2.1 Quasi-Bidirectional Ports

The Quasi-Bidirectional pins of Port 1, Port 2.6, and Port 2.7 have both input and output port configurations. They have three distinct states; low impedance current sink (Q2), low impedance current source (Q1), and high impedance current source (Q3). As a low impedance current sink, the pin has specification of sinking up to around 0.5 mA, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

Examine Figure 10. When a '1' is written to the SFR location controlling the pin, Q1 (a low impedance MOSFET pullup) is turned on for one state time, then it is turned off and the depletion pullup holds the line at a logical '1' state. The low-impedance pullup is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pullup.

While the depletion mode pullup is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used directly with switches. However if the switch option is used, certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could cause logical operations made directly on these pins to indavertently write a 0 to pins being used as inputs. In order to perform logical operations on a port where a quasi-bidirectional pin is an input, it is necessary to guarantee that the bit associated with the input pin is always a one when writing to the port.

2.2 Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground from each pin is tyically 20 mA or more. Therefore, if all 8 pins are tied to ground, 160 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

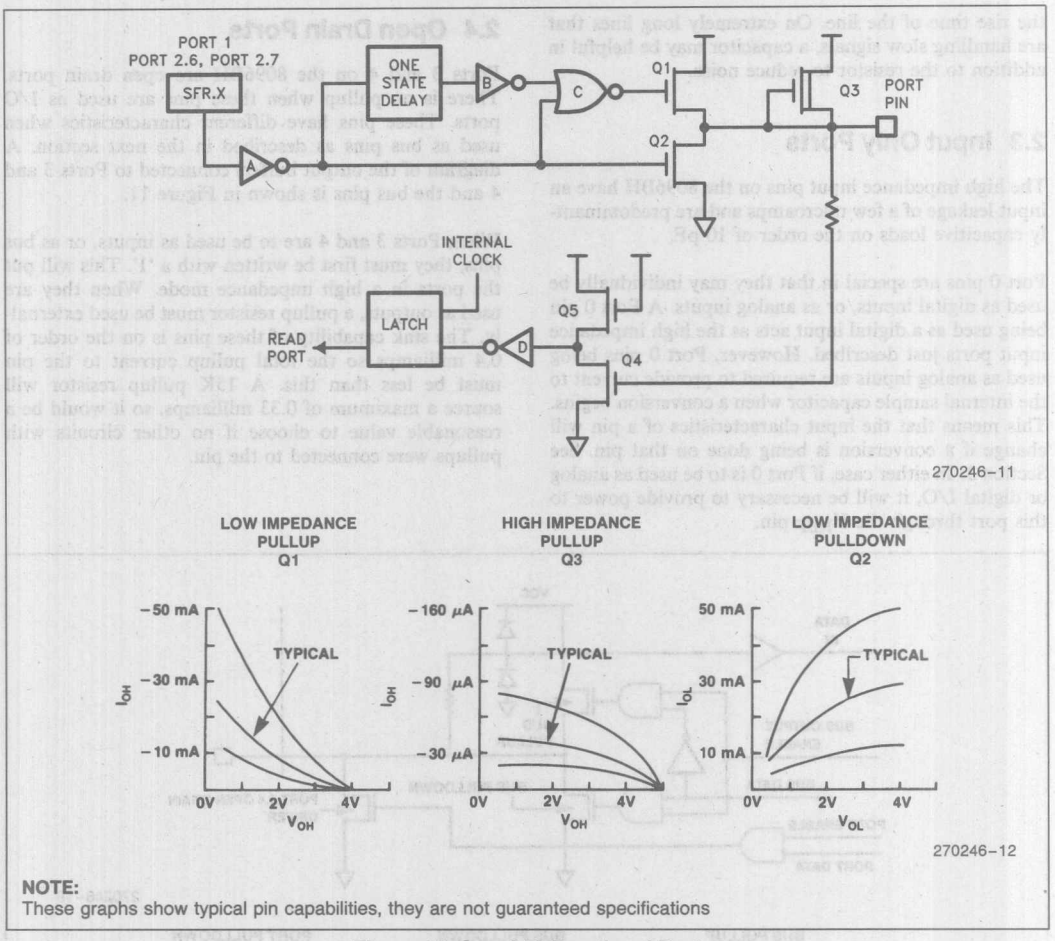
This potential problem can be solved in hardware or software. In software, never write a zero to a pin being used as an input.

In hardware, a 1K resistor in series with each pin will limit current to a reasonable value without impeding the ability to override the high impedance pullup. If all 8 pins are tied together a 120Ω resistor would be reasonable. The problem is not quite as severe when the inputs are tied to electronic devices instead of switches, as most external pulldowns will not hold 20 mA to 0.0 volts.

Writing to a Quasi-Bidirectional Port with electronic devices attached to the pins requires special attention. Consider using P1.0 as an input and trying to toggle P1.1 as an output:

```
ORB   IOPORT1, #00000001B ; Set P1.0
                          ; for input
XORB  IOPORT1, #00000010B ; Complement
                          ; P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven



NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications

Figure 10. Quasi-Bidirectional Port

high by the 8096BH it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7V. The 8096BH will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin's SFR and the pin will not toggle.

The second problem, which is related to the first, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction, then the XORB will write a zero to P1.0 and it will no longer be useable as an input.

The first situation can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works by bringing up the voltage present on the port pin. The second case can be taken care of in the software fairly easily:

```
LDB  AL, IOPORT1
XORB AL, #010B
ORB  AL, #001B
STB  AL, IOPORT1
```

A software solution to both cases is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and copy it to the port.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pullup resistor is recommended to reduce the possibility of noise glitches and to decrease

the rise time of the line. On extremely long lines that are handling slow signals, a capacitor may be helpful in addition to the resistor to reduce noise.

2.3 Input Only Ports

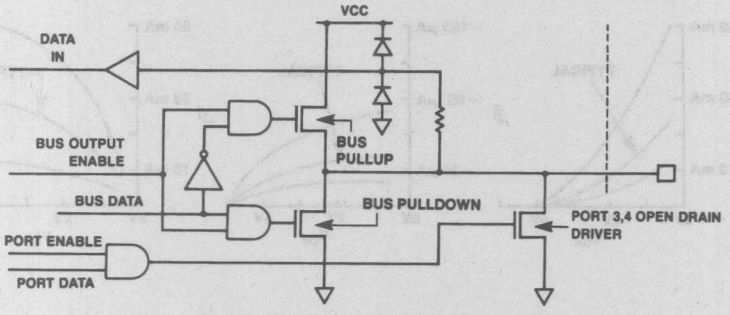
The high impedance input pins on the 8096BH have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pF.

Port 0 pins are special in that they may individually be used as digital inputs, or as analog inputs. A Port 0 pin being used as a digital input acts as the high impedance input ports just described. However, Port 0 pins being used as analog inputs are required to provide current to the internal sample capacitor when a conversion begins. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See Section 3. In either case, if Port 0 is to be used as analog or digital I/O, it will be necessary to provide power to this port through the V_{REF} pin.

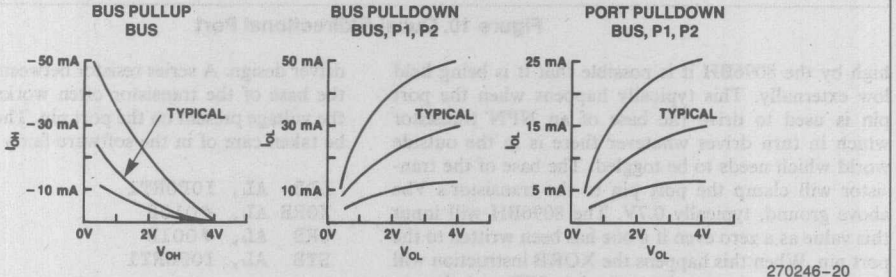
2.4 Open Drain Ports

Ports 3 and 4 on the 8096BH are open drain ports. There is no pullup when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to Ports 3 and 4 and the bus pins is shown in Figure 11.

When Ports 3 and 4 are to be used as inputs, or as bus pins, they must first be written with a '1'. This will put the ports in a high impedance mode. When they are used as outputs, a pullup resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pullup current to the pin must be less than this. A 15K pullup resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.



270246-19



270246-20

NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications.

Figure 11. Bus and Port 3 and 4 Pins

2.5 HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in Mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, and $\overline{WR}$. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide the fast rise times. When used as a low output, the pins can sink around 2 mA at 0.45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 11 shows the internal configuration of a bus pin.

3.0 ANALOG INPUTS

The on-chip A/D converter of the 8096BH can be used to digitize analog inputs while analog outputs can be generated with either the chip's PWM output or HSO unit. This section describes the analog input suggestions. See Section 4 for analog output.

The 8096BH's Integrated A/D converter includes an eight channel analog multiplexer, sample-and-hold circuit and 10-bit analog to digital converter (Figure 12). The 8096BH can therefore select one of eight analog inputs to convert, sample-and-hold the input voltage and convert the voltage into a digital value. Each conversion takes 22 microseconds, including the time required for the sample-hold (with XTAL1 = 12 MHz). The method of conversion is successive approximation.

Section 3.6 contains the definitions of numerous terms used in connection with the A/D converter.

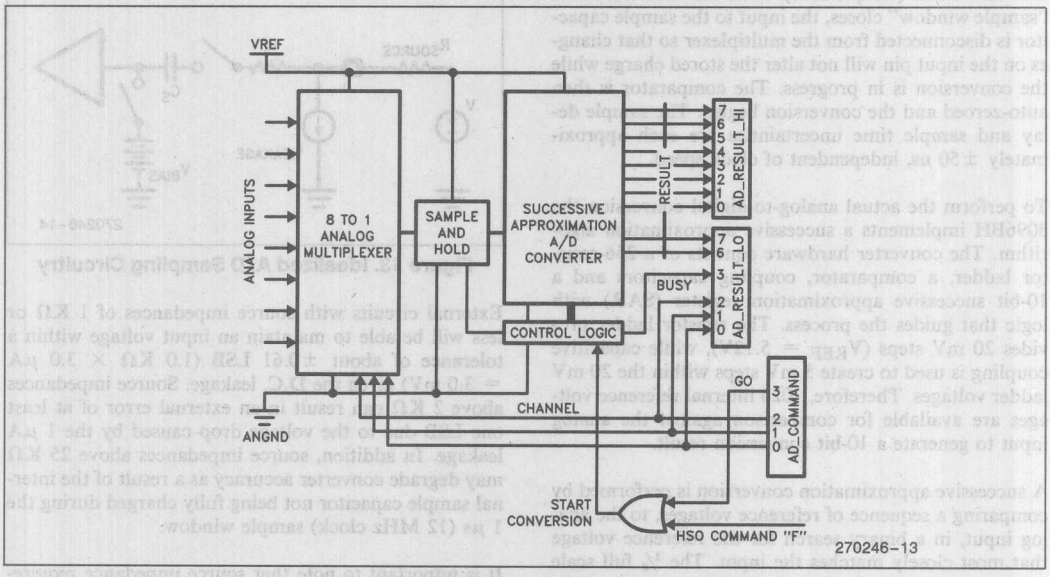


Figure 12. A/D Converter Block Diagram

3.1 A/D Overview

The conversion process is initiated by the execution of HSO command 0FH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to the A/D converter control logic. If an HSO command was used, the conversion process will begin when Timer 1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer 1 delays will occur with a variance of about ±50 ns (assuming a stable clock on XTAL1). However, conversions initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution resulting in a variance of about 0.75 μs (XTAL1 = 12 MHz).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (sample delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while the conversion is in progress. The comparator is then auto-zeroed and the conversion begins. The sample delay and sample time uncertainties are each approximately ±50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the 8096BH implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps (V_{REF} = 5.12V), while capacitive coupling is used to create 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result.

A successive approximation conversion is performed by comparing a sequence of reference voltages, to the analog input, in a binary search for the reference voltage that most closely matches the input. The ½ full scale reference voltage is the first tested. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111.1111.11b). If the analog input was less than the test voltage, bit 10 of the SAR is left a zero, and a new test voltage of ¼ full scale (0011.1111.11b) is tried. If this test voltage was lower than the analog input, bit 9 of the SAR is set and bit 8 is cleared for the next test (0101.1111.11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in the SAR where it can be read by software.

The total number of state times required is 88 for a 10-bit conversion. Attempting to short-cycle the 10-bit conversion process by reading A/D results before the done bit is set is not recommended.

3.2 A/D Interface Suggestions

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's design, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

For the 8096BH, these factors are idealized in Figure 13. The external input circuit must be able to charge a sample capacitor (C_S) through a series resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 8096BH, C_S is around 2 pF, R_I is around 5 KΩ and I_L is specified as 3 μA maximum. In determining the necessary source impedance R_S, the value of V_{BIAS} is not important.

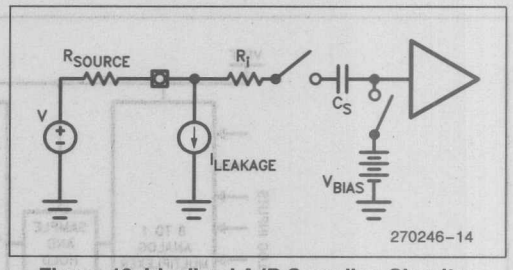


Figure 13. Idealized A/D Sampling Circuitry

External circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.61 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ can result in an external error of at least one LSB due to the voltage drop caused by the 1 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μs (12 MHz clock) sample window.

It is important to note that source impedance requirements relax if an external capacitor of sufficient size is attached directly to the analog input pin. Since the internal sample capacitor is around 2.0 pF, an external 0.005 μF capacitor (2048 × 2.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C. leakage caused by the 8096BH, 0.6 mV (less than 0.15 LSB) will be lost from a 0.005 μF capacitor in 1 μs. Therefore, the capacitor

connected externally to the pin should be at least 0.005 μF if the source impedance is too large to provide the needed accuracy on its own. However, if the external signal changes slowly, it is recommended that the largest acceptable capacitance be used, given the input signal frequency.

Placing an external capacitor on each analog input will also reduce the sensitivity to noise, as the capacitor combines with series resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to the external capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 14 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input. Should the input voltage inappropriately drop significantly below ground, diode D2 will forward bias at about 0.8 DCV. Since the specification of the pin has an absolute maximum low voltage rating of −0.3 DCV, this will leave about 0.5 DCV across the 270Ω resistor, or about 2.0 mA of current. This should limit current to a safe amount. *However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.*

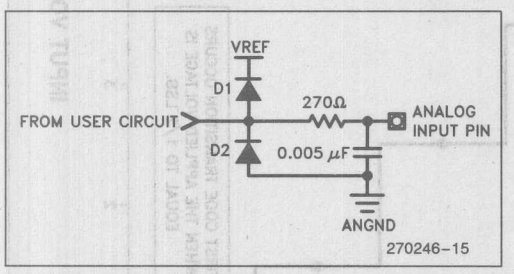


Figure 14. Suggested A/D Input Circuit

3.3 Analog References

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to the two V_{SS} pins as close to the chip as possible with minimum trace length. Bypass capacitors should also be used between V_{REF} and ANGND. ANGND should be within about a tenth of a volt V_{SS}. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. Figure 6 shows all of these connections.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC}. In addition, V_{REF} and ANGND must be connected even if the A/D converter is not being used. Remember that Port 0 receives its power from the V_{REF} and ANGND pins even when it is used as digital I/O.

3.4 The A/D Transfer Function

The conversion result is a 10-bit ratiometric representation of the input voltage, so the numerical value obtained from the conversion will be:

$$\text{INT}\,[1023 \times (V_{IN} - \text{ANGND})/(V_{REF} - \text{ANGND})].$$

This produces a stair-stepped transfer function when the output code is plotted versus input voltage (see Figure 15). The resulting digital codes can be taken as simple ratiometric information, or they can be used to provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, closing a servo-loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error; zero offset; full-scale error; differential non-linearity; and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, converter temperature drift, V_{CC} rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching and random noise should be considered. Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. However, the various sub-components of error are important in many applications. These error components are described in Section 3.5 and in the text below where ideal and actual converters are compared.

An unavoidable error simply results from the conversion of a continuous voltage to an integer digital representation. This error is called quantizing error, and is always ±0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 15 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 15 the Ideal Characteristic possesses unique qualities: it's first code transition occurs when the input voltage is 0.5 LSB; it's full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and it's code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

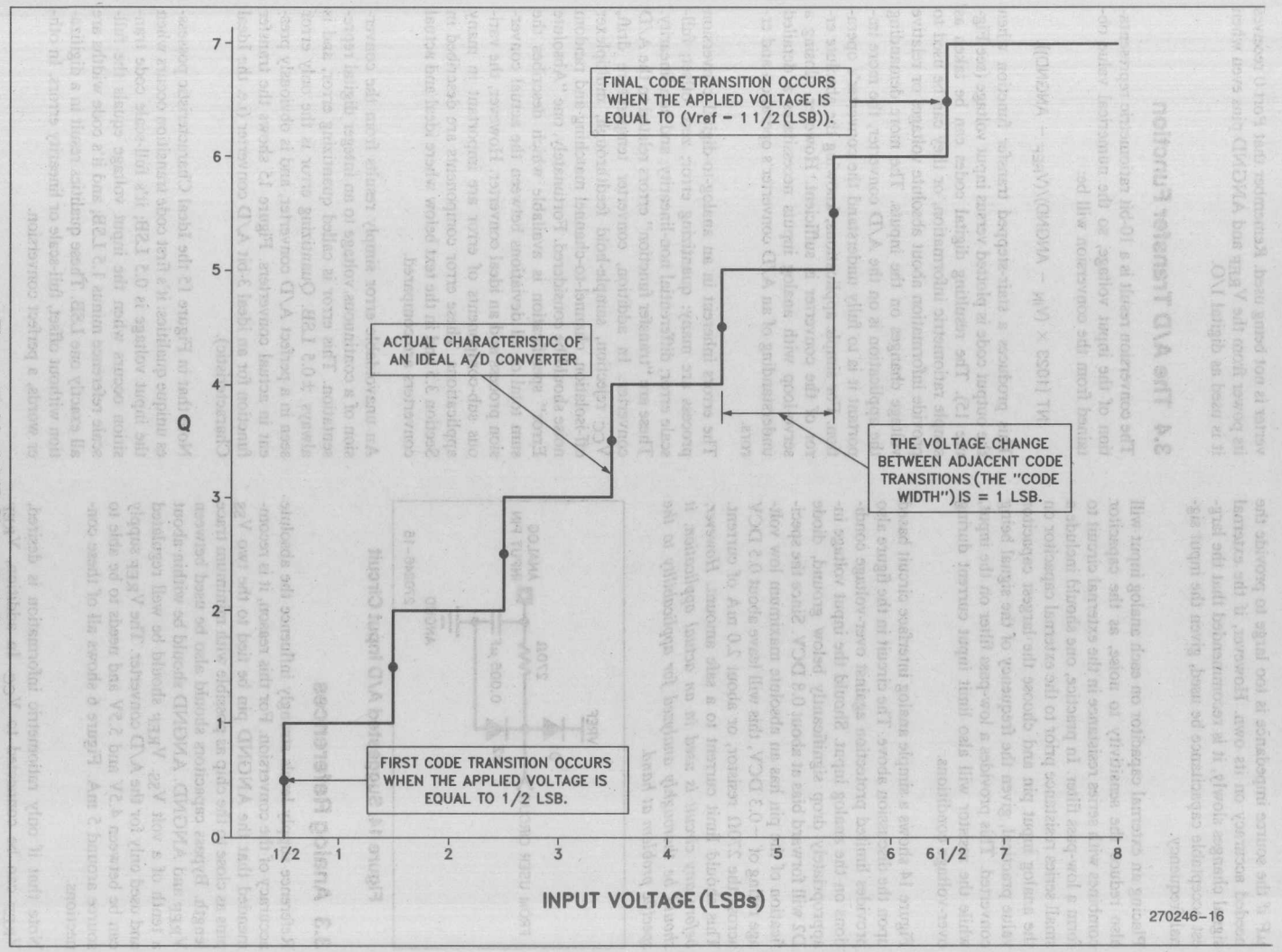


Figure 15. Ideal A/D Characteristic

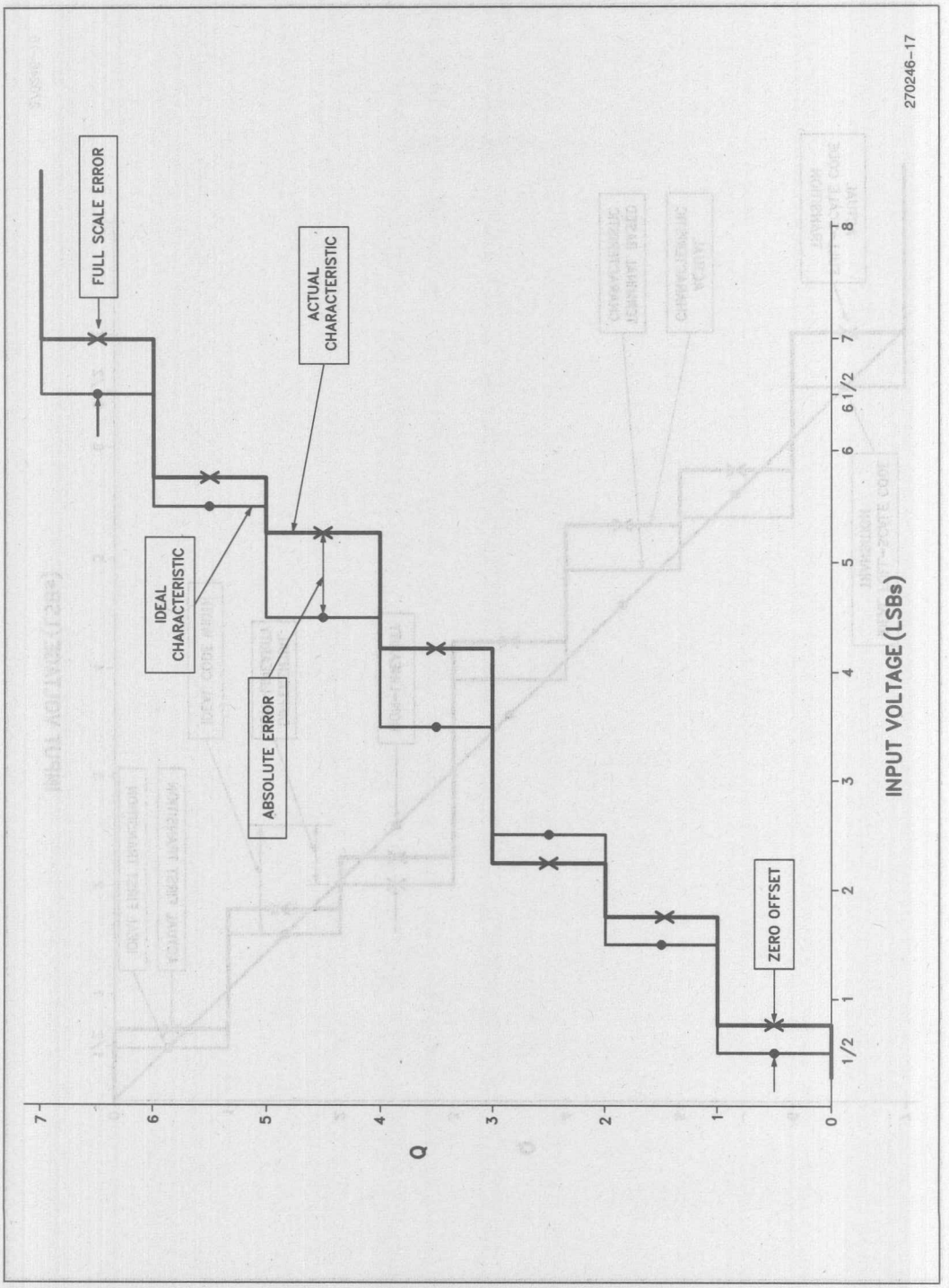


Figure 16. Actual and Ideal Characteristics

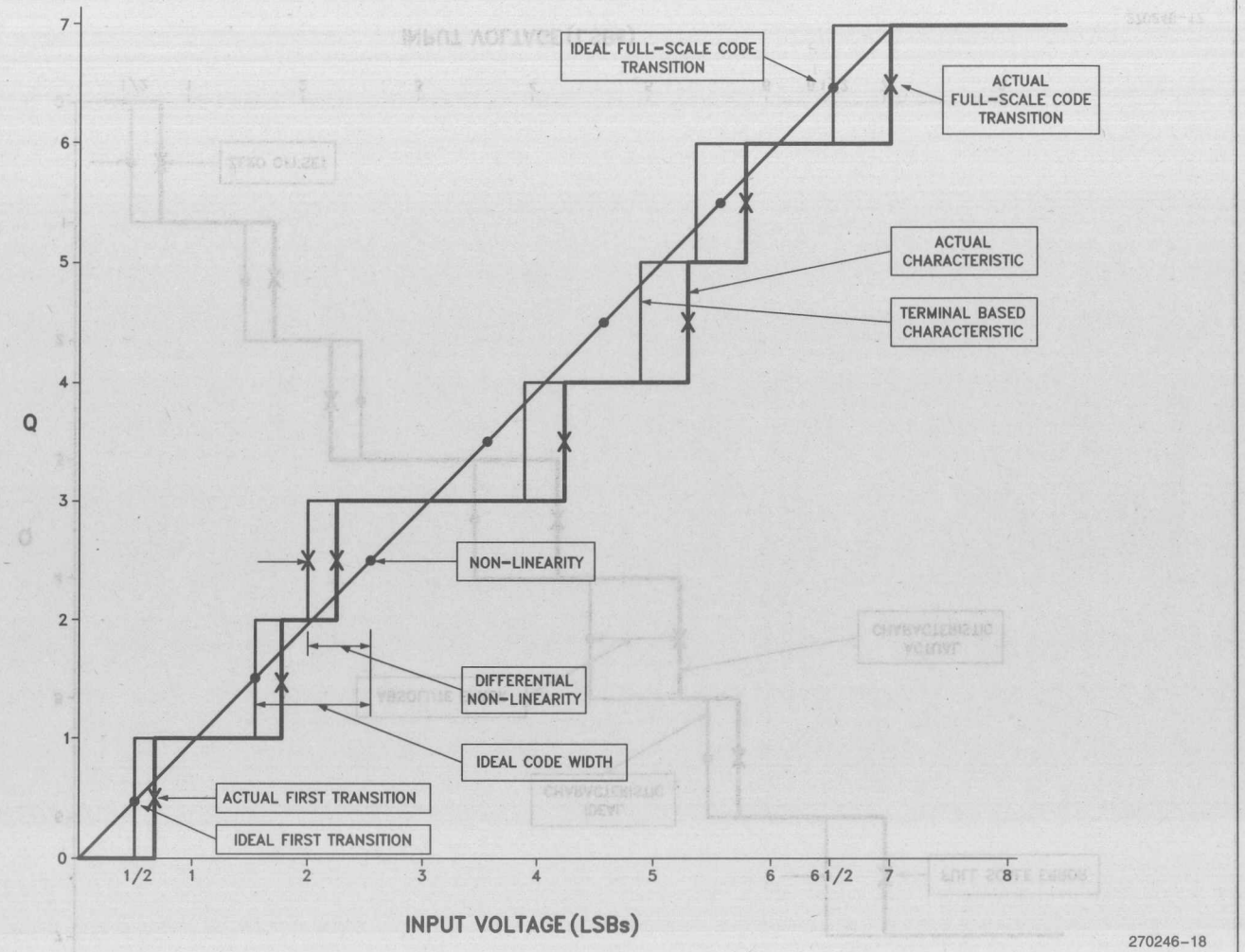


Figure 17. Terminal Based Characteristic

Figure 16 shows an Actual Characteristic of a hypothetical 3-bit converter, which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset", and the deviation of the final code transition from ideal is "full-scale error". The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measurement, whereas Non-Linearity is an overall linearity error measure.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal one LSB width. Differential Non-Linearity gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result. Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add up to produce an overall maximum departure from a linear characteristic. If the Differential Non-Linearity errors are too large, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desireable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and rotated to eliminate zero offset and full-scale error (see Figure 17). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. In addition, V_{REF} on the 8096BH could also be closely regulated and trimmed within the specified range to affect full-scale error.

Other factors that affect a real A/D Converter system include sensitivity to temperature, failure to completely reject all unwanted signals, multiplexer channel dissimilarities and random noise. Fortunately these effects are small.

Temperature sensitivities are described by the rate at which typical specifications change with a change in temperature.

Undesired signals come from three main sources. First, noise on V_{CC}—V_{CC} Rejection. Second, input signal changes on the channel being converted after the sample window has closed—Feedthrough. Third, signals applied to channels not selected by the multiplexer—Off-Isolation.

Finally, multiplexer on-channel resistances differ slightly from one channel to the next causing Channel-to-Channel Matching errors, and random noise in general results in Repeatability errors.

3.5 A/D Glossary of Terms

Figures 15, 16 and 17 display many of these terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversion under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—LEAST SIGNIFICANT BIT: The voltage value corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristics.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which as been rotated and translated to remove zero offset and full-scale error.

VCC REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

4.0 ANALOG OUTPUTS

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 18. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. Figure 19 shows two typical circuits. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 μs is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

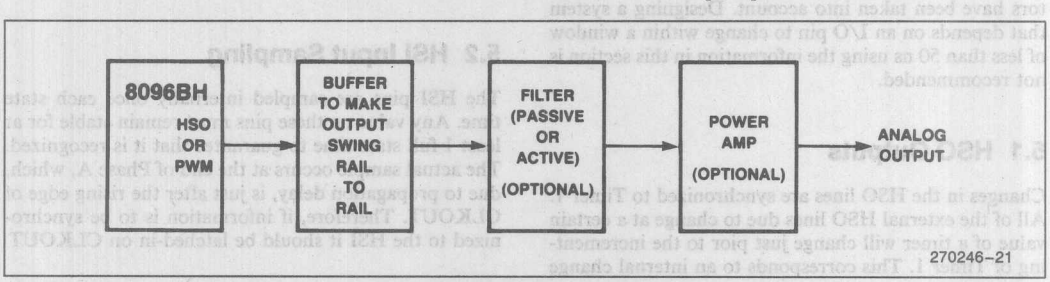


Figure 18. D/A Buffer Block Diagram

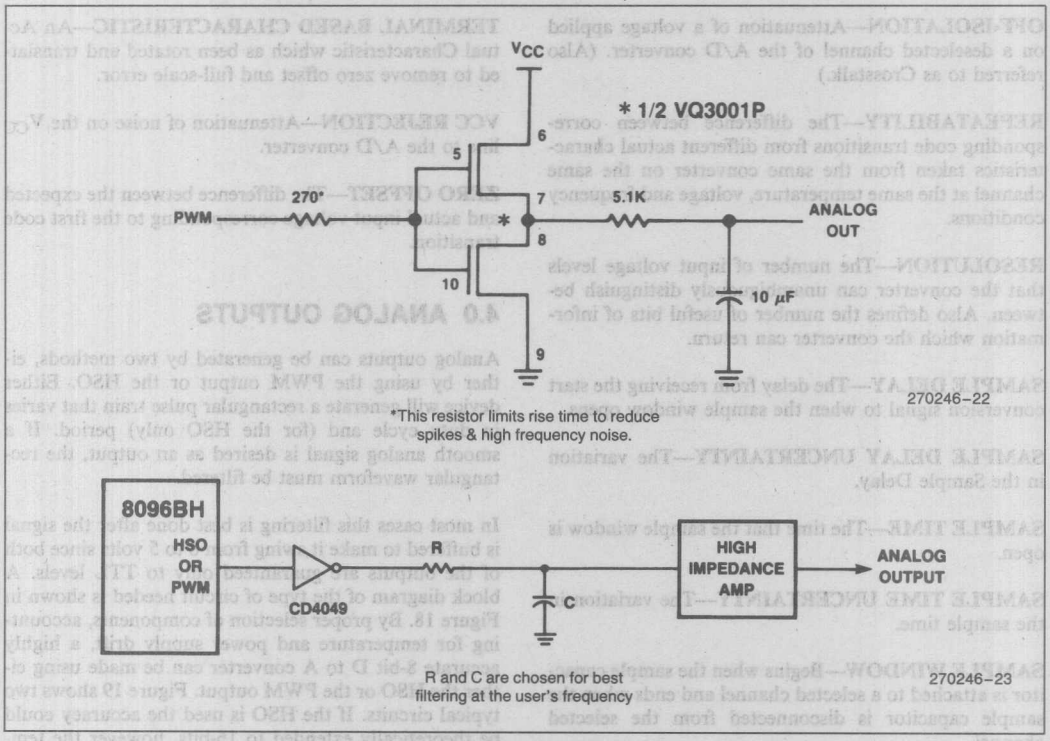


Figure 19. Buffer Circuits for D/A

5.0 I/O TIMINGS

The I/O pins on the 8096BH are sampled and changed at specific times within an instruction cycle. The changes occur relative to the internal phases shown in Figure 4. Note that the delay from XTAL1 to the internal clocks range from about 30 ns to 100 ns over process and temperature. Signals generated by internal phases are further delayed by 5 ns to 15 ns. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 ns using the information in this section is not recommended.

5.1 HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just pior to the incrementing of Timer 1. This corresponds to an internal change during Phase B every eight state times. From an external perspective the HSO pin should change just prior to the rising edge of CLKOUT and be stable by its falling edge. Information from the HSO can be latched on the CLKOUT falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

5.2 HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. The actual sample occurs at the end of Phase A, which, due to propagation delay, is just after the rising edge of CLKOUT. Therefore, if information is to be synchronized to the HSI it should be latched-in on CLKOUT

falling. The time restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (i.e. the amount of time during which Timer 1 remains constant), is stable to within about 20 ns. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

5.3 Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A/D converter. The port is sampled once every state time, however, sampling is not synchronized to Timer 1. If this port is used, the input signal on the pin must be stable one state time before the reading of the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2 and is similar to the HSI in that the sample occurs just after the rising edge of CLKOUT. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pullup will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drains, their structure is different when they are used as part of the bus. See Section 10.4 of the 8096BH Architecture chapter. Additional information on port reconstruction is available in Section 7.8 of this chapter.

6.0 SERIAL PORT TIMINGS

The serial port on the 8096BH was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096BH uses a divide by 3, the serial port on the 8096BH had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096BH so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK. Because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to $1/16$ that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 11.4 of the 8096BH Architecture chapter discusses programming the baud rate generator.

6.1 Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 20 shows the waveforms and timing. Note that the port starts functioning when a '0' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096BH by simply adding shift registers. A schematic of a typical circuit is shown in Figure 21. This circuit inverts the data coming in, so it must be reinverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

6.2 Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

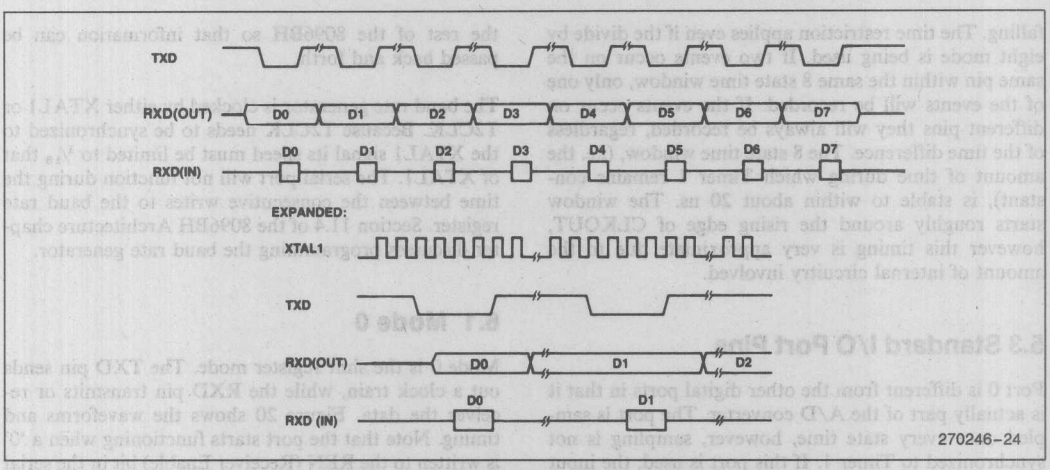


Figure 20. Serial Port Timings in Mode 0

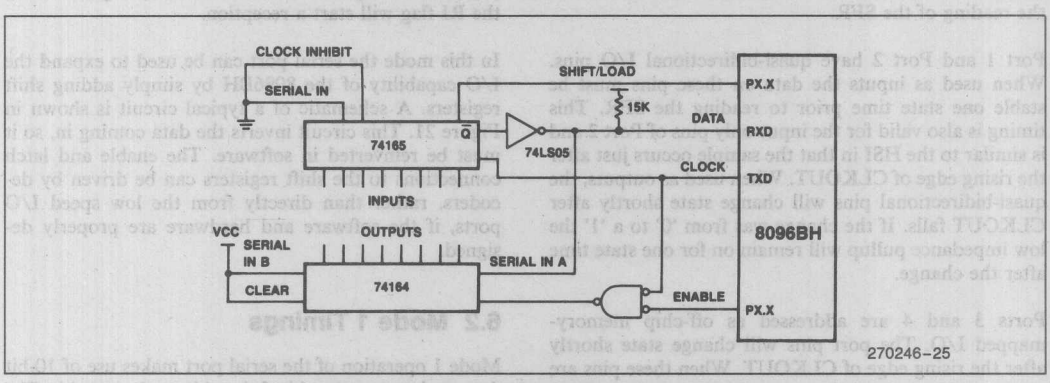


Figure 21. Mode 0 Serial Port Example

Caution should be used when using the serial port to connect more than two devices in half-duplex, (i.e. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

6.3 Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of Mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see Section 11 of the 8096BH Architecture chapter).

7.0 BUS TIMING AND MEMORY INTERFACE

7.1 Bus Functionality

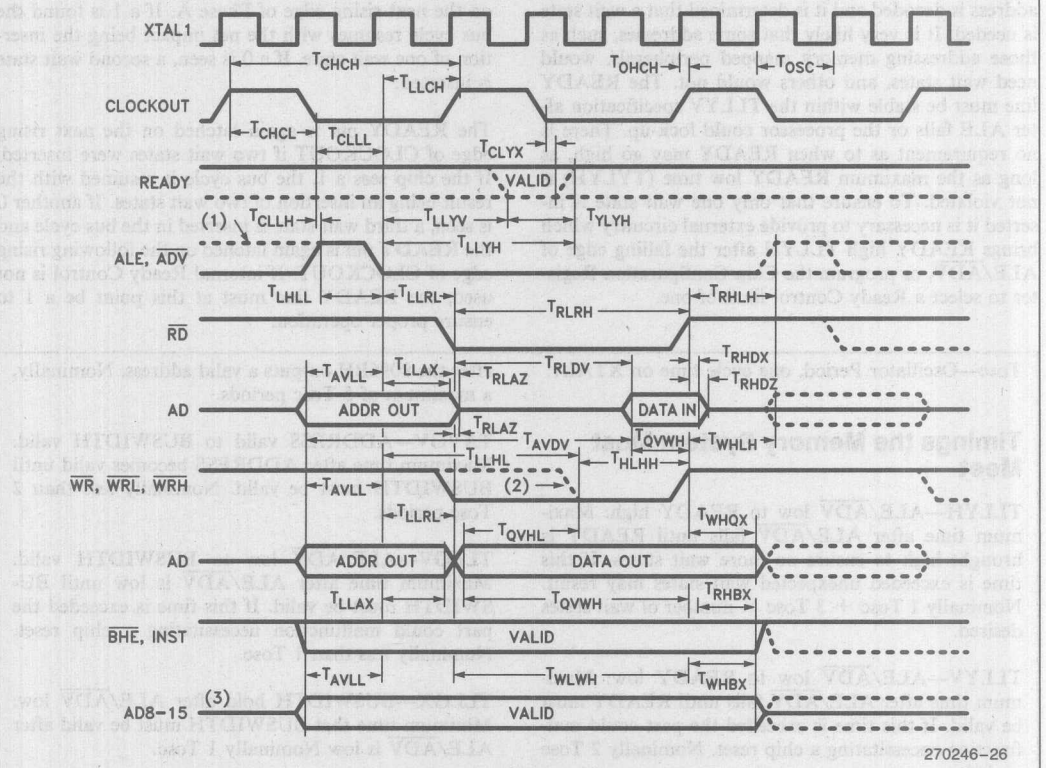
The 8096BH has a multiplexed (address/data) bus which can be dynamically configured to have an 8-bit or 16-bit data width. There are control lines to demultiplex the bus ($\overline{ALE}$ or $\overline{ADV}$), indicate reads ($\overline{RD}$), indicate writes ($\overline{WRL}$ and $\overline{WRH}$, or $\overline{WR}$ with $\overline{BHE}$ and AD0), and a signal to indicate accesses that are for an instruction fetch (INST). Section 3.5 of the 8096BH Architecture chapter contains an overview of the bus operation.

7.2 Timing Specifications

Figure 22 shows the timing of the bus signals and data lines. Please refer to the latest data sheet for the exact device you are using to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 23.

7.3 READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications, it is necessary to use the READY line to generate wait states. When the READY line is held low, the processor waits in a loop for the line to come high or until the



NOTES:
1. When ALE function is selected, the signal is always high for TLHLL. When $\overline{ADV}$ function is selected, the signal is high for at least TLHLL.
2. The dotted line applies for all 8-bit bus writes and 16-bit bus writes with the write strobe mode selected.
3. 8-bit bus only.

Figure 22. Bus Signal Timings

number of inserted wait states is equal to the limit set in the Chip Configuration Register (see Section 2 of the MCS-96 Architecture chapter). There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to provide external circuitry which brings READY high TLLYH after the falling edge of ALE/$\overline{\text{ADV}}$, or program the Chip Configuration Register to select a Ready Control limit of one.

Internally, the chip latches READY on the first falling edge of Phase A after ALE/$\overline{\text{ADV}}$ falls. Phase A is buffered and brought out externally as CLOCKOUT, so CLOCKOUT is a delayed Phase A. If a 1 is seen, the bus cycle proceeds uninterrupted with no wait state insertions. If a 0 is seen, one wait state (3 Tosc) is inserted.

If a wait state is inserted, READY is internally latched on the next rising edge of Phase A. If a 1 is found the bus cycle resumes with the net impact being the insertion of one wait state. If a 0 is seen, a second wait state is inserted.

The READY pin is again latched on the next rising edge of CLOCKOUT if two wait states were inserted. If the chip sees a 1, the bus cycle is resumed with the result being an insertion of two wait states. If another 0 is seen, a third wait state is inserted in the bus cycle and the READY pin is again latched on the following rising edge of CLOCKOUT. If internal Ready Control is not used, the READY line must at this point be a 1 to ensure proper operation.

Tosc—Oscillator Period, one cycle time on XTAL1.

Timings the Memory System Must Meet

TLLYH—ALE/$\overline{\text{ADV}}$ low to READY high: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally 1 Tosc + 3 Tosc × number of wait states desired.

TLLYV—ALE/$\overline{\text{ADV}}$ low to READY low: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 Tosc periods.

TCLYX—READY hold after CLOCKOUT low: Minimum time that the value on the READY pin must be valid after CLOCKOUT falls. The minimum hold time is always zero nanoseconds.

TYLYH—READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096BH dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV—ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096BH outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TAVGV—ADDRESS valid to BUSWIDTH valid: Maximum time after ADDRESS becomes valid until BUSWIDTH must be valid. Nominally less than 2 Tosc periods.

TLLGV—ALE/$\overline{\text{ADV}}$ low to BUSWIDTH valid: Maximum time after ALE/$\overline{\text{ADV}}$ is low until BUSWIDTH must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally less than 1 Tosc.

TLLGX—BUSWIDTH hold after ALE/$\overline{\text{ADV}}$ low: Minimum time that BUSWIDTH must be valid after ALE/$\overline{\text{ADV}}$ is low Nominally 1 Tosc.

TRLDV—$\overline{\text{READ}}$ low to DATA valid: Maximum time that the memory has to output data after $\overline{\text{READ}}$ goes low. Nominally, a maximum of 3 Tosc periods.

TRHDZ—$\overline{\text{READ}}$ high to DATA float: Time after $\overline{\text{READ}}$ is high until the memory must float the bus. The memory signal can be removed as soon as $\overline{\text{READ}}$ is not low, and must be removed within the specified maximum time from when $\overline{\text{READ}}$ is high. Nominally a maximum of 1 Tosc period.

TRHDX—DATA hold after $\overline{\text{READ}}$ goes high: Minimum time that memory must hold input DATA valid after $\overline{\text{RD}}$ is high. The hold time minimum is always zero nanoseconds.

Figure 23. Timing Specification Explanations

Timings the 8096 Will Provide

TOHCH—XTAL1 high to CLOCKOUT high: Delay from the rising edge of XTAL1 to the resultant rising edge on CLOCKOUT. Needed in systems where the signal driving XTAL1 is also used as a clock for external devices. Typically 50 to 100 nanoseconds.

TCHCH—CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL—CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH—CLKOUT low to ALE high: A help in deriving other timings. Typically plus or minus 5 ns to 10 ns.

TCLVL—CLOCKOUT low to ALE/$\overline{\text{ADV}}$ low: A help in deriving other timings. Nominally 1 Tosc.

TLLCH—ALE/$\overline{\text{ADV}}$ low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL—ALE/$\overline{\text{ADV}}$ high to ALE/$\overline{\text{ADV}}$ low: ALE/$\overline{\text{ADV}}$ high time. Useful in determining ALE/$\overline{\text{ADV}}$ rising edge to ADDRESS valid time. Nominally 1 Tosc period for ALE and 1 Tosc for $\overline{\text{ADV}}$ with back-to-back bus cycles.

TAVLL—ADDRESS valid to ALE/$\overline{\text{ADV}}$ low: Length of time ADDRESS is valid before ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX—ALE/$\overline{\text{ADV}}$ low to ADDRESS invalid: Length of time ADDRESS is valid after ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ low: Length of time after ALE/$\overline{\text{ADV}}$ falls before $\overline{\text{RD}}$ or $\overline{\text{WR}}$ fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TLLHL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time after ALE/$\overline{\text{ADV}}$ is low that the write strobe signals will go low. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 2 Tosc periods.

TRLRH—$\overline{\text{READ}}$ low to $\overline{\text{READ}}$ high: $\overline{\text{RD}}$ pulse width, nominally 1 Tosc period.

TRHLH—$\overline{\text{READ}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between $\overline{\text{RD}}$ going inactive and next ALE/$\overline{\text{ADV}}$, also used to calculate time between $\overline{\text{RD}}$ inactive and next ADDRESS valid. Nominally 1 Tosc period.

TRHBX—$\overline{\text{READ}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and BHE lines will be valid after $\overline{\text{RD}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{RD}}$ goes high. Nominally 1 Tosc.

TWHBX—$\overline{\text{WRITE}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and $\overline{\text{BHE}}$ lines will be valid after $\overline{\text{WR}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{WR}}$ goes high. Nominally 1 Tosc.

TWLWH—$\overline{\text{WRITE}}$ low to $\overline{\text{WRITE}}$ high: Write pulse width, nominally 3 Tosc periods.

THLHH—$\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ high: Write strobe signal pulse width. Nominally 2 Tosc periods.

TQVHL—OUTPUT valid to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time that OUTPUT data is valid prior to write strobes becoming active. Needed for interfacing to memories that read data on the falling edge of write. Nominally 1 Tosc.

TQVWH—OUTPUT valid to $\overline{\text{WRITE}}$ high: Time that the OUTPUT data is valid before $\overline{\text{WR}}$ is high. Nominally 3 Tosc periods.

TWHQX—$\overline{\text{WRITE}}$ high to OUTPUT not valid: Time that the OUTPUT data is valid after $\overline{\text{WR}}$ is high. Nominally 1 Tosc period.

TWHLH—$\overline{\text{WRITE}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between write high and next ALE/$\overline{\text{ADV}}$, also used to calculate the time between $\overline{\text{WR}}$ high and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 23. Timing Specification Explanations (Continued)

7.4 INST Line Usage

The INST (Instruction) line is high during bus cycles that are for an instruction fetch and low for any other bus cycle. The INST signal (not present on 48-pin versions) can be used with a logic analyzer to debug a system. In this way it is possible to determine if a fetch was for instructions or data, making the task of tracing the program much easier.

7.5 BUSWIDTH Pin Usage

The BUSWIDTH pin is a control input which determines the width of the bus access in progress. BUSWIDTH is sampled after the rising edge of the first CLOCKOUT after ALE/$\overline{\text{ADV}}$ goes low. If a one is seen, the bus access progresses as a 16-bit cycle. If a zero is seen, the bus access progresses as an 8-bit cycle. The BUSWIDTH setup and hold timing requirements appear in the data sheet.

The BUSWIDTH pin can be overridden by causing the BUS WIDTH SELECT bit in the Chip Configuration Register (CCR) to be zero. This will permanently select an 8-bit bus width. However, if the BUS WIDTH SELECT bit in the CCR is a one, the BUSWIDTH pin determines the bus width. See Section 3.5 of the 8096BH Architecture chapter. Since the BUSWIDTH pin is not available on 48-pin parts, the BUS WIDTH SELECT bit in the CCR determines bus width.

7.6 Address Decoding

The multiplexed bus of the 8096BH must be demultiplexed before it can be used. This can be done with two 74LS373 transparent latches for an 8096BH in 16-bit bus mode, or one 74LS373 for an 8096BH in 8-bit bus mode. As explained in Section 3.5 of the 8096BH Architecture chapter, the latched address signals will be referred to as MA0 through MA15 (Memory Address), and the data lines will be called MD0 through MD15 (Memory Data).

Since the 8096BH can make accesses to memory for either bytes or words, it is necessary to have a way of determining the type of access desired when the bus is 16-bits wide. For write cycles, the signals Write Low ($\overline{\text{WRL}}$) and Write High ($\overline{\text{WRH}}$) are provided. $\overline{\text{WRL}}$ will go low during all word writes and during all byte writes to an even location. Similarly, $\overline{\text{WRH}}$ will go low during all word writes and during all byte writes to an odd location. During read cycles, an 8096BH in 16-bit bus mode will always do a word read of an even location. If only one byte of the word is needed, the chip discards the byte it does not need.

Since 8096BH memory accesses over an 8-bit wide bus are always bytes, only one write strobe is needed for write cycles. For this purpose the $\overline{\text{WRL}}$ signal was made to go low for all write cycles during 8-bit bus accesses. When a word operation is requested, the bus controller performs two byte-wide bus cycles.

In many cases it may be desirable to have a write signal with a longer pulse width than $\overline{\text{WRL}}$/$\overline{\text{WRH}}$. The Write ($\overline{\text{WR}}$) line of the 8096BH is an alternate control signal that shares a pin with $\overline{\text{WRL}}$ and is only available in 16-bit bus mode. $\overline{\text{WR}}$ is nominally one Tosc longer than the $\overline{\text{WRL}}$/$\overline{\text{WRH}}$ signals, but goes low for any write cycle. Therefore it is necessary to decode for the type of write (byte or word) desired.

The Byte High Enable ($\overline{\text{BHE}}$) signal and MA0 can be used for this purpose. $\overline{\text{BHE}}$ is an alternate control sig-

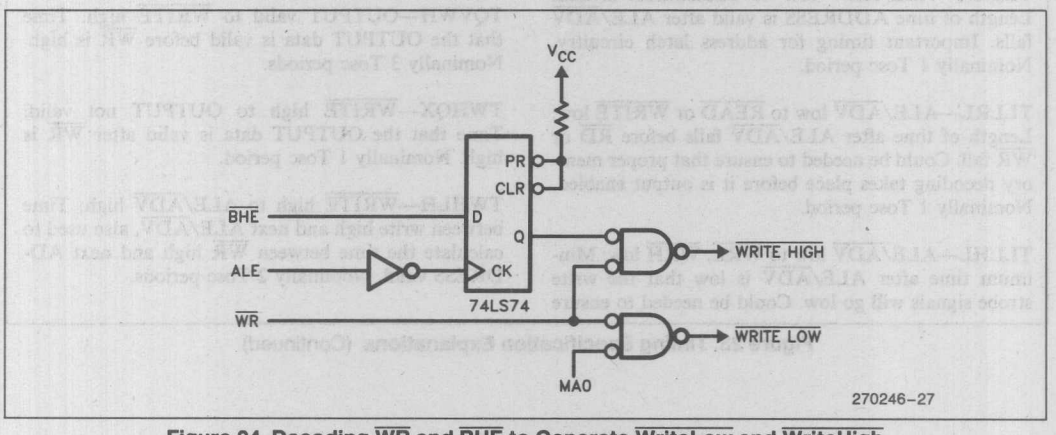


Figure 24. Decoding $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ to Generate $\overline{\text{WriteLow}}$ and $\overline{\text{WriteHigh}}$

nal that shares a pin with $\overline{WRH}$. When $\overline{BHE}$ is low, the high byte of the 16-bit bus is enabled. When MA0 is low, the lower byte is enabled. When MA0 is low and $\overline{BHE}$ is low, both bytes are enabled. Figure 24 shows how to use $\overline{WR}$, $\overline{BHE}$ and $\overline{MA0}$ to decode bus accesses. It's important to note that this decoding inserts a delay in the write signal which must be considered in a system timing analysis.

External memory systems for the 8096BH can be set up in many ways. Figures 25 through 28 show block diagrams of memory systems using an 8-bit bus with a single EPROM, using an 8-bit bus with RAM and EPROM, using a 16-bit bus with two external EPROMs and using a 16-bit bus in a RAM and ROM system.

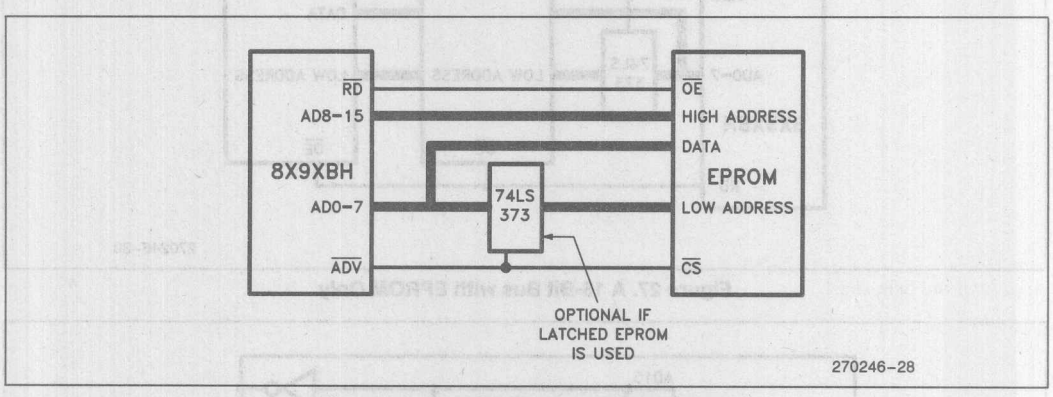


Figure 25. An 8-Bit Bus with EPROM Only

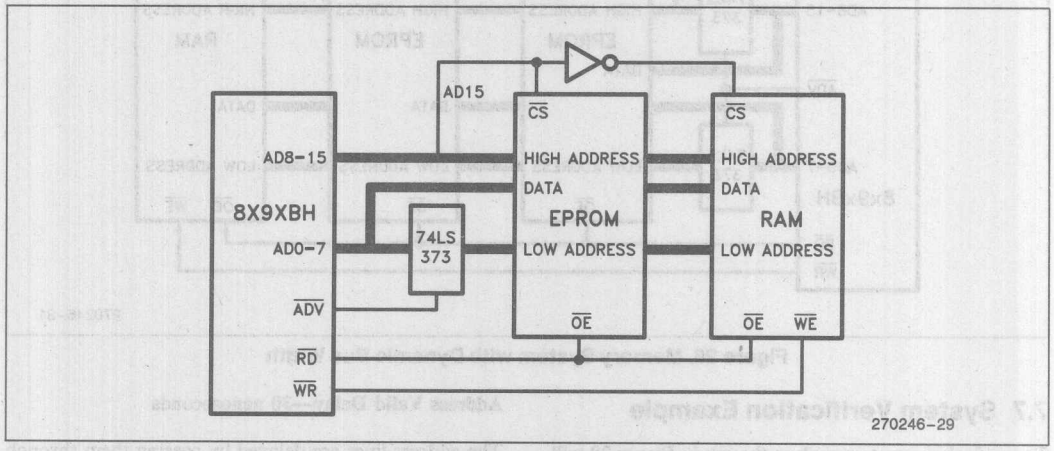


Figure 26. An 8-Bit Bus with EPROM and RAM

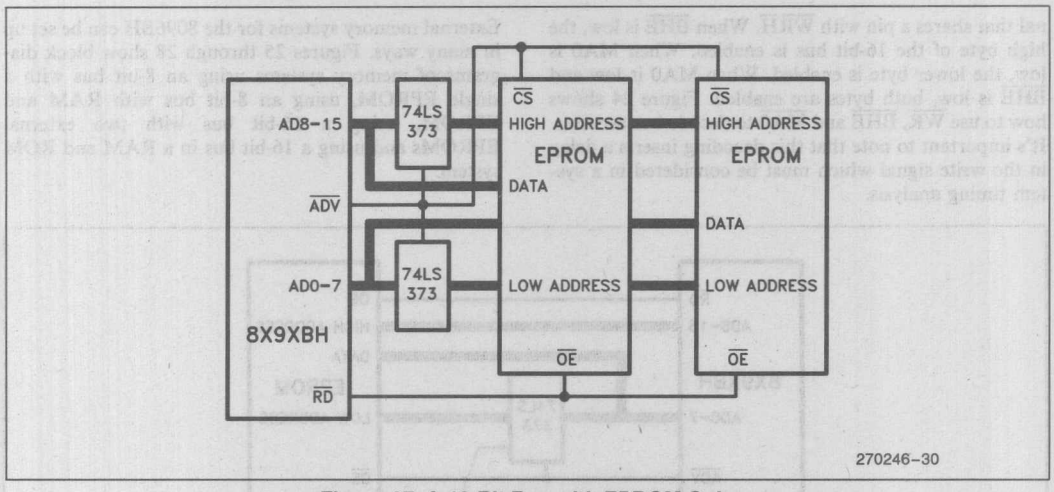


Figure 27. A 16-Bit Bus with EPROM Only

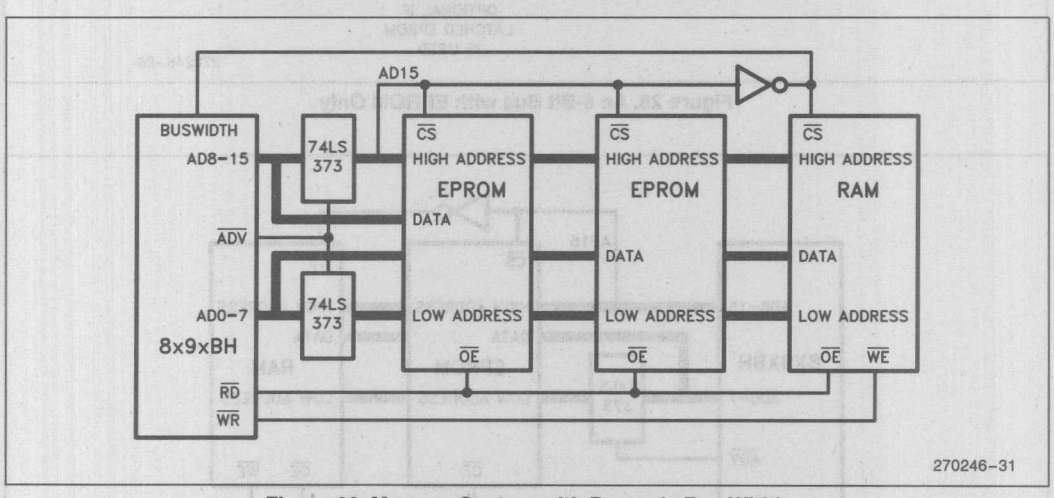


Figure 28. Memory System with Dynamic Bus Width

7.7 System Verification Example

To verify that a system such as the one in Figure 29 will work with the 8096BH, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using representative 8096BH specifications. These specifications will be different for each part number and temperature range, so the results of this example must be modified based on the most recent data sheet for the specific part to be used.

The timings of signals that the processor and memory use are affected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

Address Valid Delay—30 nanoseconds

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18 ns after Address is valid or 30 ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE

Minimum ALE pulse width = Tosc − 25 (TLHLL)

Minimum Addr set-up to ALE falling = Tosc − 25 (TAVLL)

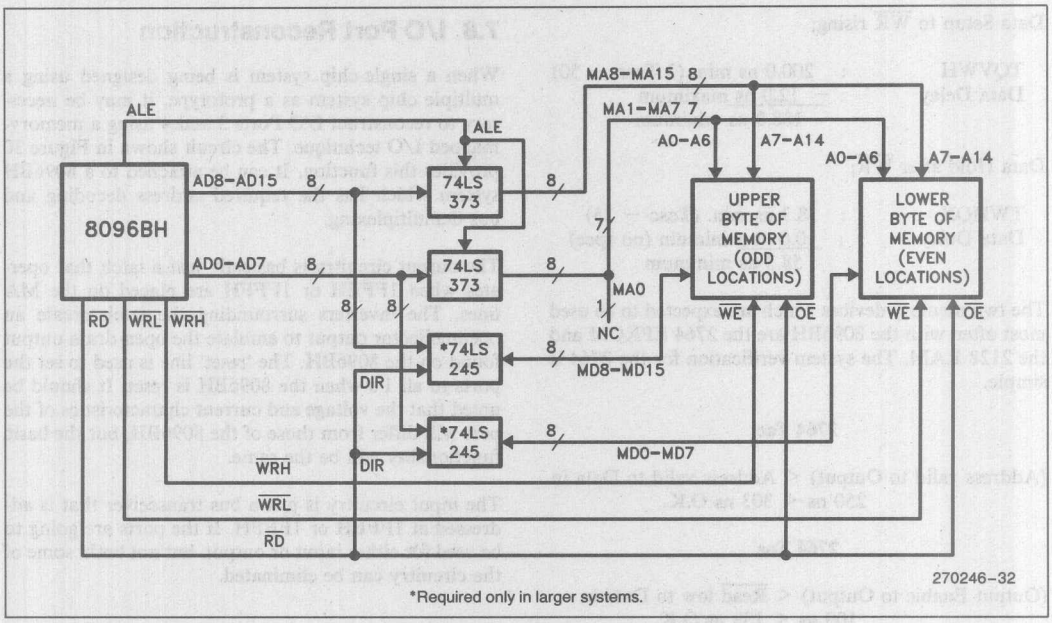


Figure 29. RAM/ROM Memory System

Therefore, in the worst case, ALE would occur 0 ns before Address valid.

Total delay from 8096BH Address stable to MA (Memory Address) stable would be:

ALE delay from address	− 0
74LS373 clock to output	30
	30 nanoseconds

If Limited by Address Valid

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controled by ALE and has a value of 30 nanoseconds.

Delay of Data Transfer to/from Processor—12 nanoseconds

The $\overline{RD}$ low to Data valid specification (TRLDV) is 3 Tosc − 50, (200 ns at 12 MHz). The 74LS245 is enabled by $\overline{RD}$ and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled for write, except during a read, so there is no enable delay to consider for write operations.

The Data In to Data Out delay of the 74LS245 is 12 ns.

CHARACTERISTICS OF A 12 MHz 8096BH SYSTEM WITH LATCHES

Required by system:

Address valid to Data in;

TAVDV	:	345.6 ns max. (5 Tosc − 70)
Address Delay	:	− 30.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		303.6 ns maximum

Read low to Data in:

TRLDV	:	200.0 ns max. (3 Tosc − 50)
Address Delay	:	− 00.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		188.0 ns maximum

Provided by system:

Address valid to Control;

TLLRL	:	63.3 ns min. (Tosc − 20)
TAVLL	:	58.3 ns min. (Tosc − 25)
Address Delay	:	− 30.0 ns maximum
$\overline{WR}$ Delay	:	− 00.0 ns minimum
		91.6 ns minimum

Write Pulse Width;

THLHH	:	146.6 ns min. (2 Tosc − 20)
		146.6 ns minimum

Data Setup to $\overline{WR}$ rising;

TQVWH : 200.0 ns min. (3 Tosc − 50)
Data Delay : − 12.0 ns maximum
188.0 ns minimum

Data Hold after $\overline{WR}$;

TWHQX : 58.3 ns min. (Tosc − 25)
Data Delay : 0.0 ns minimum (no spec)
58.3 ns minimum

The two memory devices which are expected to be used most often with the 8096BH are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 303 ns O.K.

2764 Toe

(Output Enable to Output) < $\overline{Read}$ low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the $\overline{RD}$ (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 303 ns O.K.

2128-20 Toe < $\overline{Read}$ low to Data in
65 ns < 188 ns O.K.

The write calculation are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width
100 ns < 146 ns O.K.

2128 Tds (Data Setup) < Data Setup to $\overline{WR}$ rising
65 ns < 188 ns O.K.

2128 Tdh (Data Hold) < Data Hold after $\overline{WR}$
0 ns < 58 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 29. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

7.8 I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O Ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 30 provides this function. It can be attached to a 8096BH system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096BH. The 'reset' line is used to set the ports to all 1's when the 8096BH is reset. It should be noted that the voltage and current characteristics of the port will differ from those of the 8096BH, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

8.0 NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the PC board to find itself in the path of electrostatic discharges. Glitches and noise on the PC board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096BH has a Watchdog Timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096BH instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

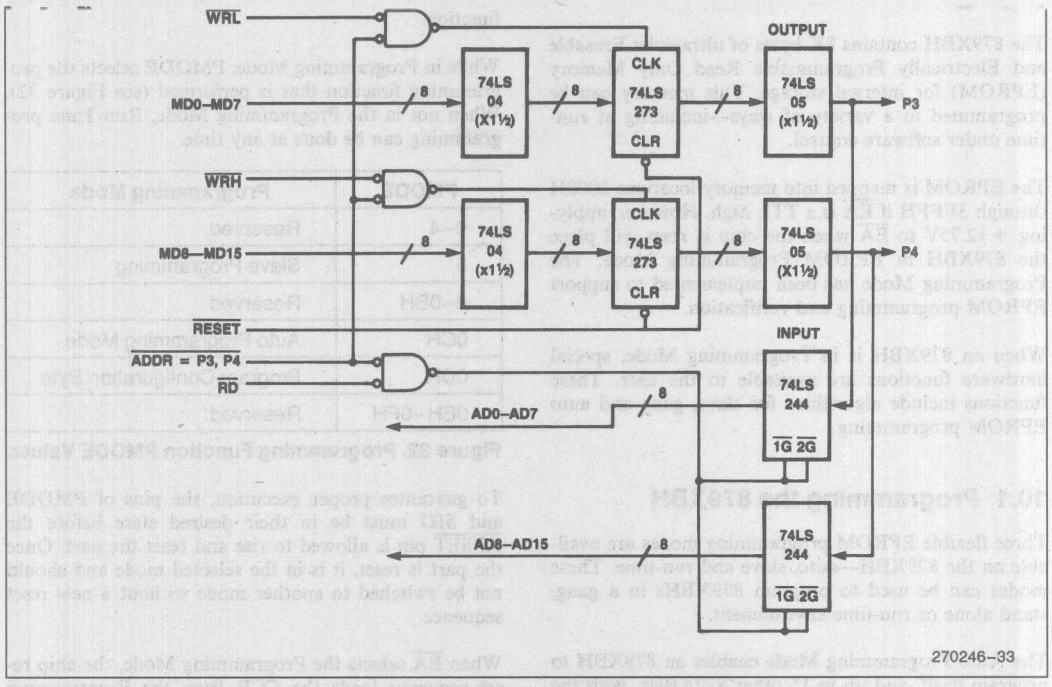


Figure 30. I/O Port Reconstruction

Many hardware solutions exist for keeping PC board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper PC board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Application Note AP-125, "Designing Microcontroller Systems for Noisy Environments".

9.0 PACKAGING

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without on-chip ROM/EPROM and with or without an A/D converter. A summary of the available options is shown in Figure 31.

The 48-pin versions are available in ceramic and plastic 48-pin Dual-In-Line package (DIP). The ceramic versions have part numbers with the prefix "C". The plastic versions have the prefix "P".

The 68-pin versions are available in a ceramic pin grid array (PGA), a plastic leaded chip carrier (PLCC) and a Type B leadless chip carrier (LCC). PGA devices have part numbers with the prefix "C". PLCC devices have the prefix "N". LCC devices have the prefix "R".

Specifications for the various members of the MCS-96 family are contained in the next chapter.

	ROMless		With ROM		With EPROM	
	68-pin	48-pin	68-pin	48-pin	68-pin	48-pin
Without A to D	8096		8396		8796	
With A to D	8097	8095	8397	8395	8797	8795

Figure 31. The MCS®-96 Family of Products

10.0 EPROM PROGRAMMING

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

10.1 Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 33 shows how the pins are renamed. Figure 34 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 32). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 32. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCB is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCB must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

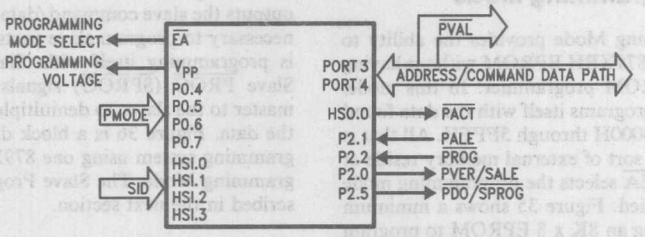


Figure 33. Programming Mode Pin Function

Name	Function
PMODE	**PROGRAMMING MODE SELECT:** Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	**SLAVE ID NUMBER:** Used to assign each slave pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	**PROGRAMMING ALE INPUT**: Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	**PROGRAMMING PULSE:** Accepted by 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	**PROGRAMMING ACTIVE:** Used in the Auto-Programming Mode to indicate when programming activity is complete.
PVER	**PROGRAM VERIFIED:** A signal output after a programming operation by parts in the Slave Programming Mode and after programming in the Auto Configuration Byte Programming Mode. This signal is on Port 2.0 and is asserted as a logic 1 if the bytes program correctly.
$\overline{\text{PVAL}}$	**PROGRAM VALID:** These signals indicate the success or failure of programming in the Auto Programming Mode and when using this mode for gang programming. For the Auto Programming Mode this signal is asserted at Port 3.0. When using this mode for gang programming, all bits of Port 3 and Port 4 are asserted to indicate programming validity of the various slaves. A zero indicates successful programming.
$\overline{\text{PDO}}$	**PROGRAMMING DURATION OVERFLOWED:** A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	**SLAVE ALE:** Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	**SLAVE PROGRAMMING PULSE:** Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indictates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	**ADDRESS/COMMAND/DATA BUS:** Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Each line should be pulled up to VCC through a resistor.

Figure 34. Programming Mode Pin Definitions

10.2 Auto Programming Mode

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that $\overline{EA}$ selects the programming mode and that VPP is applied. Figure 35 shows a minimum configuration for using an 8K x 8 EPROM to program one 879XBH in the Auto Programming Mode.

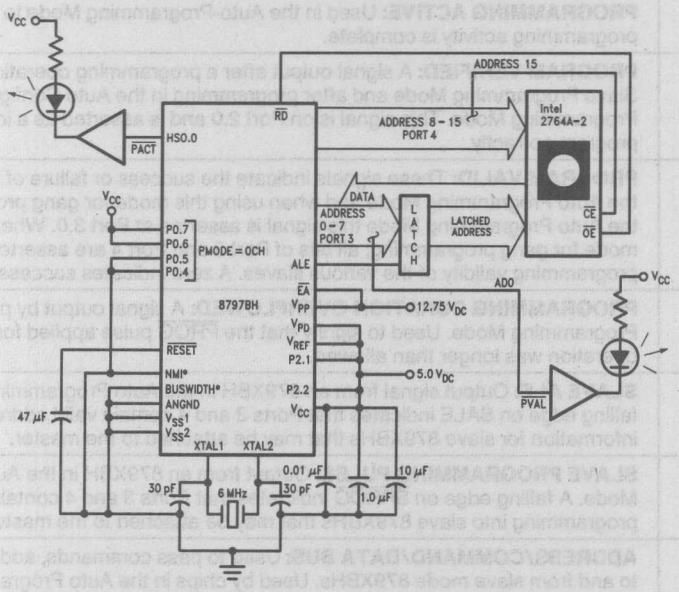
The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on Port 3.0 ($\overline{PVAL}$) if it programmed correctly and a 1 if it failed.

10.2.1 GANG PROGRAMMING WITH THE AUTO PROGRAMMING MODE

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave $\overline{PROG}$ ($\overline{SPROG}$) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 36 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

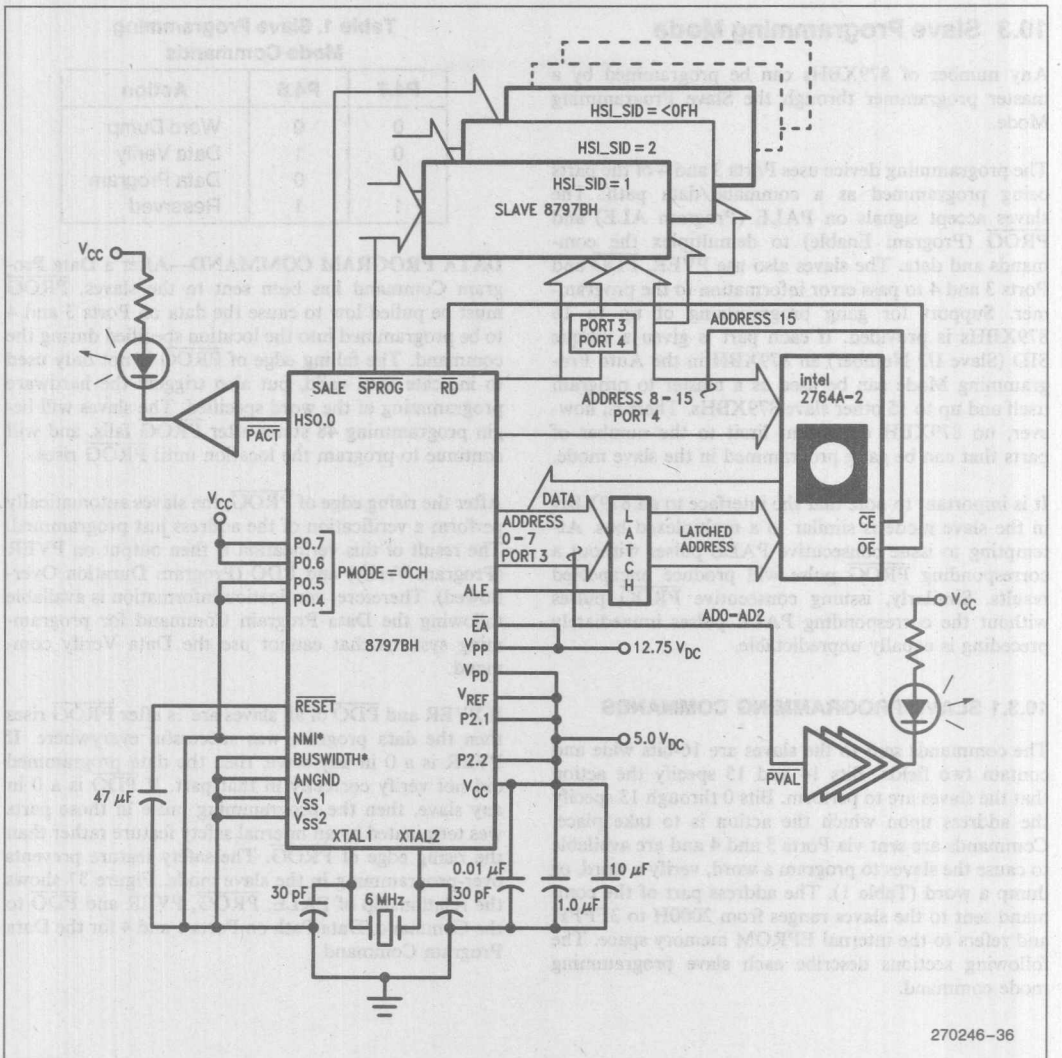
The master 879XBH first reads a word from the external memory controlled by ALE, $\overline{RD}$ and $\overline{WR}$. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on $\overline{SPROG}$. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete $\overline{PACT}$ goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.



NOTE:
Ports 3 and 4 should have pullups to VCC.

Figure 35. The Auto Programming Mode



270246-36

NOTE:
$\overline{EA}$ and VPP on slaves must be at +12.75 Vdc. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to VCC. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

Figure 36. Gang Programming with the Auto Programming Mode

10.3 Slave Programming Mode

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding PALE pulses immediately preceding is equally unpredictable.

10.3.1 SLAVE PROGRAMMING COMMANDS

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 1). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each slave programming mode command.

Table 1. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 37 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

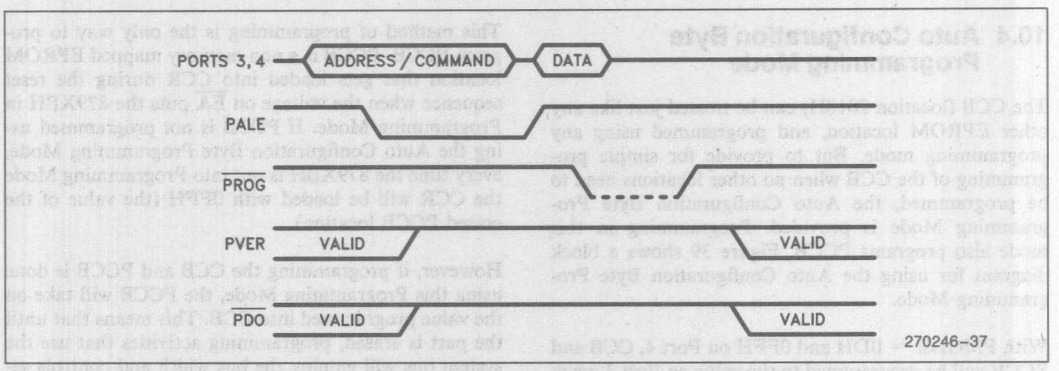


Figure 37. Data Program Signals in Slave Programming Mode

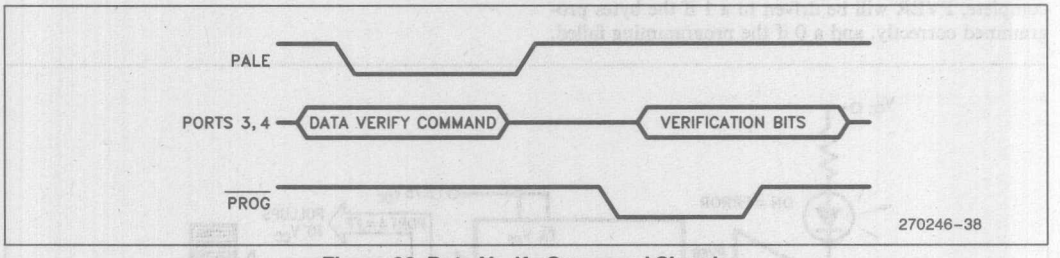


Figure 38. Data Verify Command Signals

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 and 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{\text{PROG}}$ from the programmer governs when the slaves drive the bus. Figure 38 shows the relationship of Ports 3 and 4 to PALE and $\overline{\text{PROG}}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND—When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{\text{PROG}}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 22.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

10.3.2 GANG PROGRAMMING WITH THE SLAVE PROGRAMMING MODE

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{\text{PDO}}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{\text{PDO}}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the Auto Programming Mode could be the master programmer if 15 or fewer slaves need to be programmed (see Gang Programming with the Auto Programming Mode).

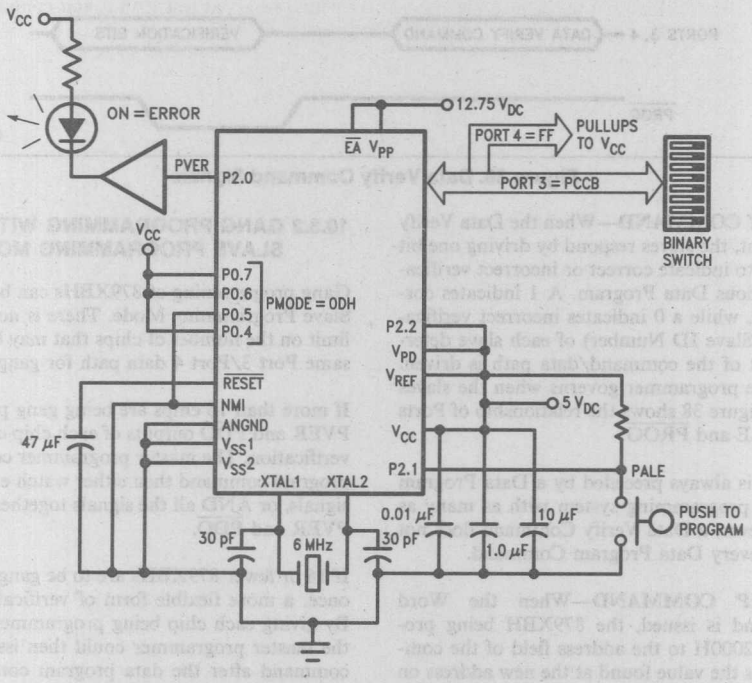
10.4 Auto Configuration Byte Programming Mode

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 39 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and 0FFH on Port 4, CCB and PCCB will be programmed to the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on $\overline{EA}$ puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming the CCB and PCCB is done using this Programming Mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system bus will employ the bus width and controls selected by the user's CCB.



NOTES:
1. Tie Port 3 to the value desired to be programmed into CCB, and PCCB.
2. Make all necessary minimum connections for power, ground and clock.

Figure 39. The Auto CCR Programming Mode

10.5 Run-Time Programming

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to VPP. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation—internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few mircoseconds. An intelligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming Algorithm be implemented.

After the programming of a location has started, care must be taken to ensure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 40 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self". The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:
    POP  temp                          ;take parameters from the
                                        STACK
    POP  address_temp
    POP  data_temp
    PUSH temp

    PUSHF                              ;save current status
    LDB  int_mask , #enable_swt_only   ;enable only swt interrups
    LDB  HSO_COMMAND , #SWTO_ovf       ;load swt command to interrupt
    ADD  HSO_TIME,TIMER1, #program_pulse ;when program pulse time
                                        ;has elapsed
    EI
    ST   data-temp, [address_temp]
    CALL 201AH
    POPF
    RET

SWT_ISR:
    . . .

swtO_expired:
    POP  0
    RET
    . . .
```

Figure 40. Programming the EPROM from Internal Memory Execution

10.6 ROM/EPROM Program Lock

Protection mechanisms have been provided on the ROM and EPROM versions of the 8096BH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

10.6.1 LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable VPP to the EPROM. This protects the entire EPROM 2000H–3FFFH from inadvertant or unauthorized programming, and also prevents writes to the EPROM from upsetting program execution. If write protection is not enabled, a data write to an internal EPROM location will begin programming that location, and continue programming the location until a data read of the internal EPROM is executed. While programming, instruction fetches from internal EPROM will not be successful.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H–3FFFH if the slave program counter is in the range 2000H–3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even though the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

10.6.2 AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110B).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H–202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H–11FFH and 4000H–5FFFH. Unspecified data appears at the low addresses.

Internal EPROM/ROM is dumped to 4000H–5FFFH, beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:
Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

10.7 Modified Quick-Pulse Programming™ Algorithm

The Modified Quick-Pulse Programming Algorithm calls for each EPROM location to receive 25 separate 100 μs (±5 μs) program cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with V_{PP} = 12.75V ±0.25V and V_{CC} = 5.0V ±0.5V.

10.8 Signature Word

The 8X9XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 2. 8X9XBH Signature Words

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

10.9 Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

10.7 Modified Quick-Pulse Programming™ Algorithm

The Modified Quick-Pulse Programming Algorithm for each EPROM location to receive 25 separate 100 ms (±5 %) program cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 87/83XH parts is done with V_{PP} = 12.75 ±0.25V and V_{CC} = 5.0V ±0.5V.

10.8 Signature Word

The 8X9XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 2. 8X9XBH Signature Words

Device	Signature Word
87/83XH	89CFH
85/81XH	89CEH
80/51XH	Undefined

10.9 Erasing the 87/83XH EPROM

Initially, and after each erasure, all bits of the 87/83XH are in the "1" state. Data is introduced by selectively

80C196KB User's Guide

3

intel

October 1988

80C196KB User's Guide

Order Number: 270651-001

80C196KB USER'S GUIDE

CONTENTS

PAGE

1.0 CPU OPERATION 3-4
- 1.1 Memory Controller 3-5
- 1.2 CPU Control 3-5
- 1.3 Internal Timing 3-5

2.0 MEMORY SPACE 3-7
- 2.1 Register File 3-7
- 2.2 Special Function Registers 3-7
- 2.3 Reserved Memory Spaces 3-11
- 2.4 Internal ROM and EPROM 3-11
- 2.5 System Bus 3-12

3.0 SOFTWARE OVERVIEW 3-12
- 3.1 Operand Types 3-12
- 3.2 Operand Addressing 3-13
- 3.3 Program Status Word 3-15
- 3.4 Instruction Set 3-17
- 3.5 80C196KB Instruction Set Additions and Differences 3-23
- 3.6 Software Standards and Conventions 3-23
- 3.7 Software Protection Hints 3-24

4.0 PERIPHERAL OVERVIEW 3-24
- 4.1 Pulse Width Modulation Output (D/A) 3-25
- 4.2 Timer1 and Timer2 3-25
- 4.3 High Speed Inputs (HSI) 3-26
- 4.4 High Speed Outputs (HSO) 3-26
- 4.5 Serial Port 3-26
- 4.6 A/D Converter 3-28
- 4.7 I/O Ports 3-28
- 4.8 Watchdog Timer and Clock Failure Detect 3-28

5.0 INTERRUPTS 3-29
- 5.1 Interrupt Control 3-31
- 5.2 Interrupt Priorities 3-31
- 5.3 Critical Regions 3-33
- 5.4 Interrupt Timing 3-33
- 5.5 Interrupt Summary 3-34

6.0 Pulse Width Modulation Output (D/A) 3-35
- 6.1 Analog Output 3-37

CONTENTS

PAGE

7.0 TIMERS 3-38
- 7.1 Timer1 3-38
- 7.2 Timer2 3-38
- 7.3 Sampling on External Timer Pins 3-38
- 7.4 Timer Interrupts 3-39

8.0 HIGH SPEED INPUTS 3-39
- 8.1 HSI Modes 3-40
- 8.2 HSI Status 3-41
- 8.3 HSI Interrupts 3-42
- 8.4 HSI Input Sampling 3-42

9.0 HIGH SPEED OUTPUTS 3-42
- 9.1 HSO Interrupts and Software Timers 3-43
- 9.2 HSO CAM 3-44
- 9.3 HSO Status 3-45
- 9.4 Clearing the HSO and Locked Entries 3-45
- 9.5 HSO Precautions 3-46
- 9.6 PWM Using the HSO 3-46
- 9.7 HSO Output Timing 3-49

10.0 SERIAL PORT 3-49
- 10.1 Serial Port Status and Control 3-49
- 10.2 Serial Port Interrupts 3-50
- 10.3 Serial Port Modes 3-51
- 10.4 Multiprocessor Communications 3-53

11.0 A/D CONVERTER 3-53
- 11.1 A/D Conversion Process 3-55
- 11.2 A/D Interface Suggestions 3-55
- 11.3 The A/D Transfer Function 3-56
- 11.4 A/D Glossary of Terms 3-60

12.0 I/O PORTS 3-61
- 12.1 Input Ports 3-62
- 12.2 Quasi-Bidirectional Ports 3-62
- 12.3 Output Ports 3-64
- 12.4 Ports 3 and 4/AD0–15 3-64

CONTENTS

PAGE

13.0 MINIMUM HARDWARE CONSIDERATIONS 3-65
- 13.1 Power supply 3-65
- 13.2 Noise Protection Tips 3-66
- 13.3 Oscillator and Internal Timings 3-66
- 13.4 Reset and Reset Status 3-68
- 13.5 Minimum Hardware Connections 3-69

14.0 SPECIAL MODES OF OPERATION 3-71
- 14.1 Idle Mode 3-71
- 14.2 Powerdown Mode 3-71
- 14.3 ONCE and Test Modes 3-72

15.0 MEMORY CONTROLLER AND THE MEMORY MAP 3-72
- 15.1 Memory Controller 3-73
- 15.2 Memory Map and Reserved Locations 3-73

16.0 EXTERNAL MEMORY INTERFACING 3-74
- 16.1 Bus Operation 3-74
- 16.2 Chip Configuration Register 3-75
- 16.3 Bus Width 3-78
- 16.4 $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ Protocol 3-79
- 16.5 AC Timing Explanations 3-81
- 16.6 Memory System Examples 3-84
- 16.7 I/O Port Reconstruction 3-86

17.0 USING ROM AND EPROM PARTS 3-86
- 17.1 ROM/EPROM Memory Protection Options 3-86
- 17.2 Programming the 87C196KB 3-86
- 17.3 Auto Programming Mode 3-88
- 17.4 Slave Programming Mode 3-89

The 80C196KB family is a CHMOS branch of the MCS®-96 family. Other members of the MCS-96 family include the 8096-90, 8096BH and 8098. All of the MCS-96 components share a common instruction set and architecture. However the CHMOS components have enhancements to provide higher performance at lower power consumptions. To further decrease power usage, these parts can be placed into idle and power-down modes.

MCS-96 family members are all high-performance microcontrollers with a 16-bit CPU and at least 230 bytes of on-chip RAM. They are register-to-register machines, so no accumulator is needed, and most operations can be quickly performed from or to any of the registers. In addition, the register operations can control the many peripherals which are available on the chips. These peripherals include a serial port, A/D converter, PWM output, up to 48 I/O lines and a High-Speed I/O subsystem which has 2 16-bit timer/counters, an 8-level input capture FIFO and an 8-entry programmable output generator.

Typical applications for MCS-96 products are closed-loop control and mid-range digital signal processing. MCS-96 products are being used in modems, motor controls, printers, engine controls, photocopiers, anti-lock brakes, air conditioner temperature controls, disk drives, and medical instrumentation.

There are many members of the 80C196KB family, so to provide easier reading this manual will refer to the 80C196KB family generically as the 80C196KB. Where information applies only to specific components it will be clearly indicated.

The 80C196KB can be separated into four sections for the purpose of describing its operation. A block diagram is shown in Figure 1-1. There is the CPU and architecture, the instruction set, the peripherals and the bus unit. Each of the sections will be sub-divided as the discussion progresses. Let us first examine the CPU.

1.0 CPU OPERATION

The major components of the CPU on the 80C196KB are the Register File and the Register/Arithmetic Logic Unit (RALU). Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU does not use an accumulator. Instead, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, absence of accumulator bottleneck, and fast throughput and I/O times.

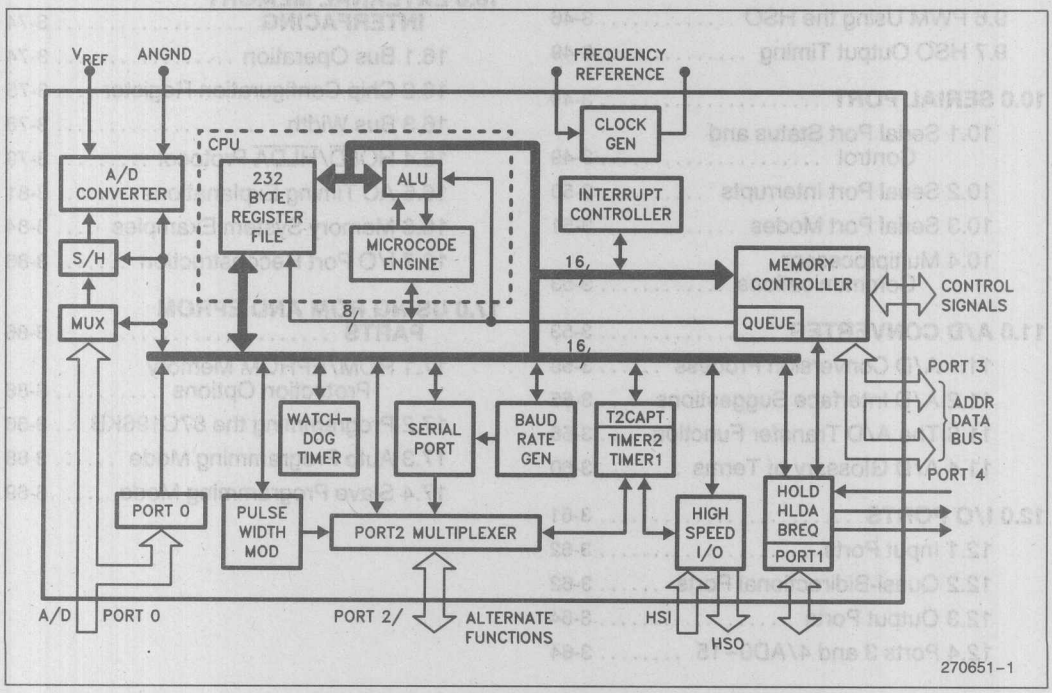


Figure 1-1. 80C196KB Block Diagram

The CPU on the 80C196KB is 16 bits wide and connects to the interrupt controller and the memory controller by a 16-bit bus. In addition, there is an 8-bit bus which transfers instruction bytes from the memory controller to the CPU. An extension of the 16-bit bus connects the CPU to the peripheral devices.

1.1 Memory Controller

The RALU talks to the memory, except for the locations in the register file and SFR space, through the memory controller. Within the memory controller is a bus controller, a four byte queue and a Slave Program Counter (Slave PC). Both the internal ROM/EPROM bus and the external memory bus are driven by the bus controller. Memory access requests to the bus controller can come from either the RALU or the queue, with queue accesses having priority. Requests from the queue are always for data at the address in the slave PC.

By having program fetches from memory referenced to the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address sequence changes because of a jump, interrupt, call or return, the slave PC is loaded with a new value, the queue is flushed, and processing continues.

Execution speed is increased by using a queue since it usually keeps the next instruction byte available. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Section 3 show the normal execution times with no wait states added and the 16-bit bus selected. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in the table.

When debugging code using a logic analyzer, one must be aware of the queue. It is not possible to determine when an instruction will begin executing by simply watching when it is fetched, since the queue is filled in advance of instruction execution.

1.2 CPU Control

A microcode engine controls the CPU, allowing it to perform operations with any byte, word or double word in the 256 byte register space. Instructions to the CPU are taken from the queue and stored temporarily in the instruction register. The microcode engine decodes the instructions and generates the correct sequence of events to have the RALU perform the desired function. Figure 1-2 shows the memory controller, RALU, instruction register and the control unit.

REGISTER/ALU (RALU)

Most calculations performed by the 80C196KB take place in the RALU. The RALU, shown in Figure 1-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementor is used for the Program Counter (PC) as it accesses operands. However, PC changes due to jumps, calls, returns and interrupts must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" and "Upper Word" are used together for the 32-bit instructions and as temporary registers for many instructions. Repetitive shifts are counted by the 6-bit "Loop Counter".

A third temporary register stores the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

Several constants, such as 0, 1 and 2 are stored in the RALU to speed up certain calculations. (e.g. making a 2's complement number or performing an increment or decrement instruction.) In addition, single bit masks for bit test instructions are generated in the constant register based on the 3-bit Bit Select register.

1.3 Internal Timing

The 80C196KB requires an input clock on XTAL1 to function. Since XTAL1 and XTAL2 are the input and output of an inverter a crystal can be used to generate the clock. Details of the circuit and suggestions for its use can be found in Section 13.

Internal operation of the 80C196KB is based on the crystal or external oscillator frequency divided by 2. Every 2 oscillator periods is referred to as one "state time", the basic time measurement for all 80C196KB operations. With a 12 MHz oscillator, a state time is 167 nanoseconds. With an 8 MHz oscillator, a state time is 250 nanoseconds, the same as an 8096 running with a 12 MHz oscillator. Since the 80C196KB will be run at many frequencies, the times given throughout this chapter will be in state times or "states", unless otherwise specified. A clock out (CLKOUT) signal,

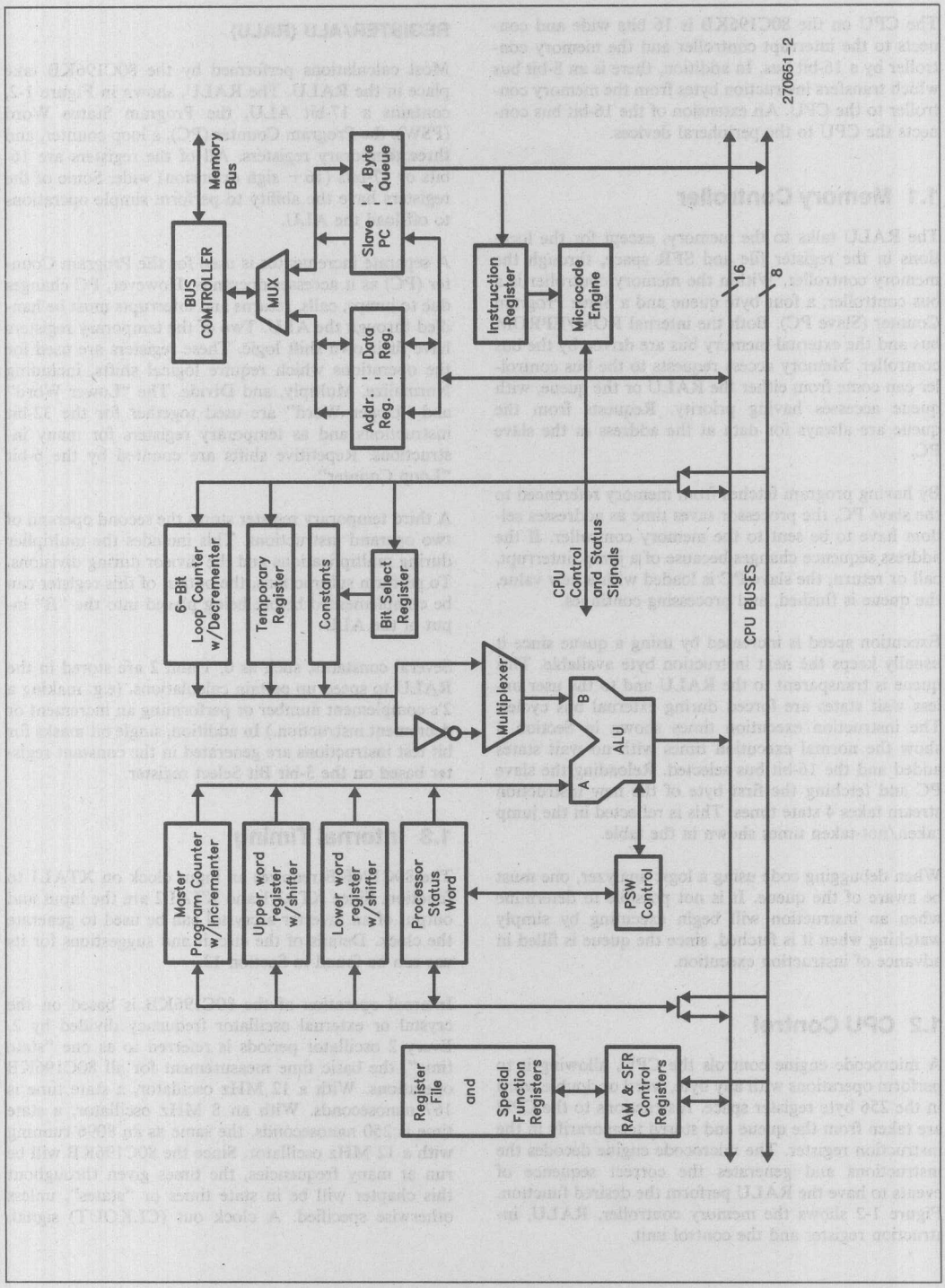


Figure 1-2. RALU and Memory Controller Block Diagram

shown in Figure 1-3, is provided as an indication of the internal machine state. Details on timing relationships can be found in Section 13.

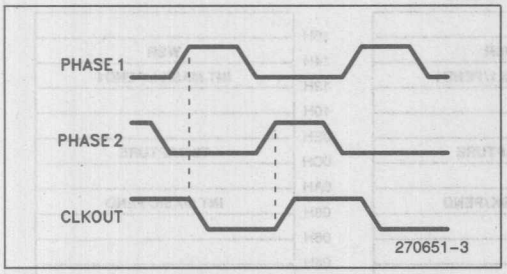


Figure 1-3. Internal Clock Waveforms

2.0 MEMORY SPACE

The addressable memory space on the 80C196KB consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2080H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in Figure 2-1.

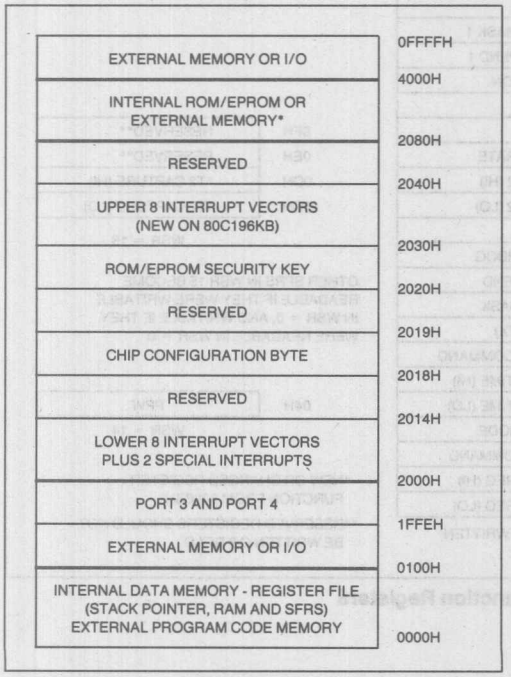


Figure 2-1. 80C196KB Memory Map

2.1 Register File

Locations 00H through 0FFH contain the Register File and Special Function Registers, (SFRs). The RALU can operate on any of these 256 internal register locations, but code can not be executed from them. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools

The internal RAM from location 018H (24 decimal) to 0FFH is the Register File. It contains 232 bytes of RAM which can be accessed as bytes (8 bits), words (16 bits), or double-words (32 bits). Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". This memory region, as well as the status of the majority of the chip, is kept intact while the chip is in the Powerdown Mode. Details on Powerdown Mode are discussed in Section 14.

Locations 18H and 19H contain the stack pointer. These are not SFRs and may be used as standard RAM if stack operations are not being performed. Since the stack pointer is in this area, the RALU can easily operate on it. The stack pointer must be initialized by the user program and can point anywhere in the 64K memory space. Operations to the stack cause it to build down, so the stack pointer should be initialized to 2 bytes above the highest stack location, and must contain word (even) addresses.

2.2 Special Function Registers

Locations 00H through 17H are the I/O control registers or SFRs. All of the peripheral devices on the 80C196KB (except Ports 3 and 4) are controlled through these registers. As shown in Figure 2-2, three SFR windows are provided on the 80C196KB.

Switching between the windows is done using the Window Select Register (WSR) at location 14H in all of the windows. The PUSHA and POPA instructions push and pop the WSR so it is easy to change between windows. Only three values may be written to the WSR, 0, 14 and 15. Other values are reserved for use in future parts and will cause unpredictable operation.

Window 0, the register window selected with WSR = 0, is a superset of the one used on the 8096. As depicted in Figure 2-3, it has 24 registers, some of which have different functions when read than when written. Registers which are new to the 80C196KB or have changed functions from the 8096 are indicated in the figure.

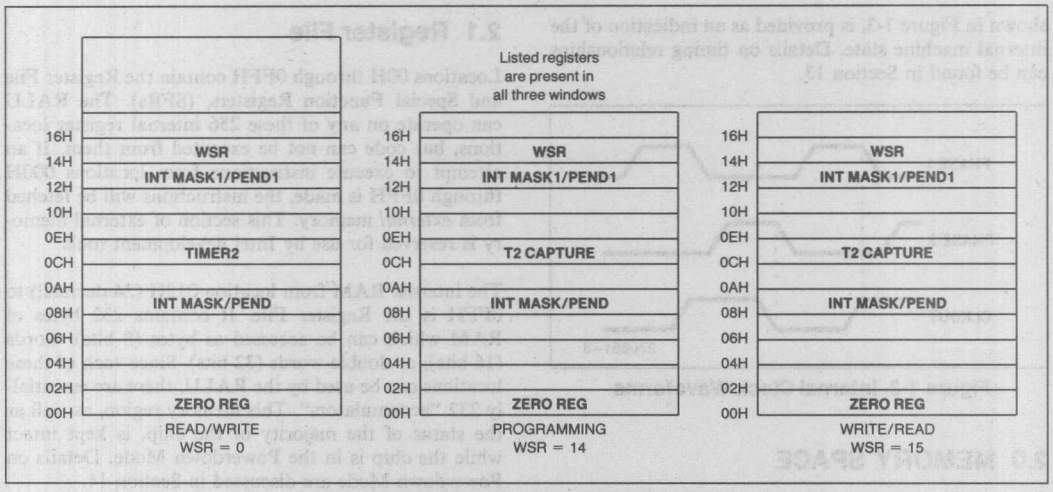


Figure 2-2. Multiple Register Windows

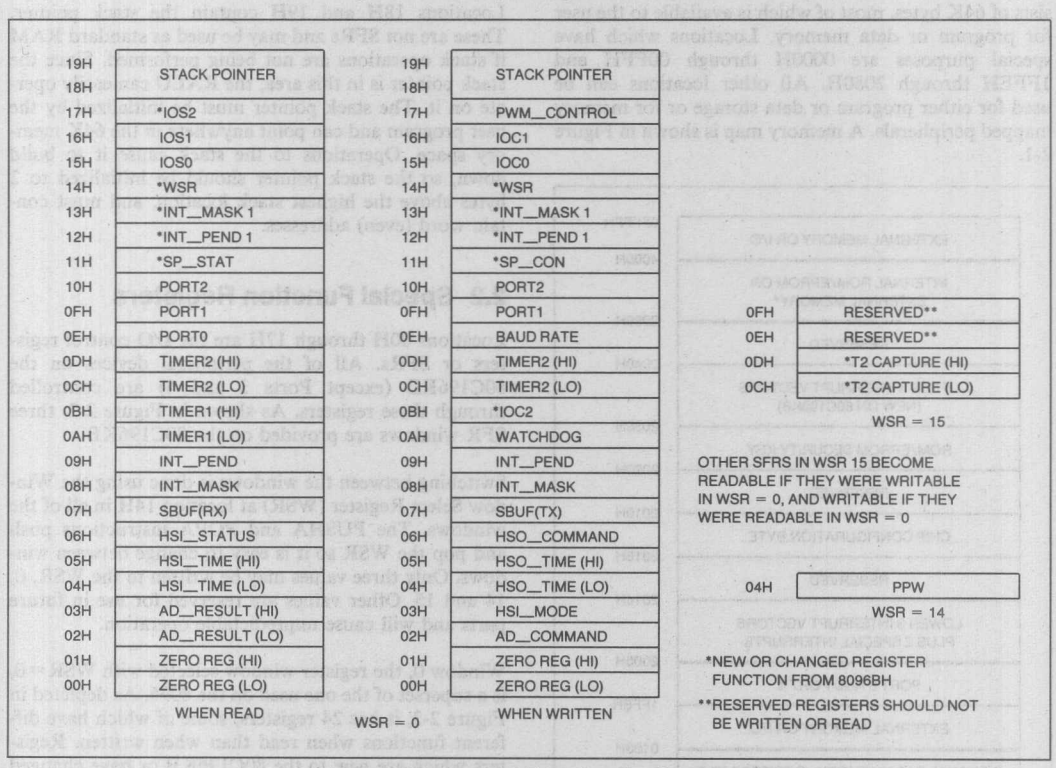


Figure 2-3. Special Function Registers

Register	Description
R0	Zero Register - Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.
AD__RESULT	A/D Result Hi/Low - Low and high order results of the A/D converter
AD__COMMAND	A/D Command Register - Controls the A/D
HSI__MODE	HSI Mode Register - Sets the mode of the High Speed Input unit.
HSI__TIME	HSI Time Hi/Lo - Contains the time at which the High Speed Input unit was triggered.
HSO__TIME	HSO Time Hi/Lo - Sets the time or count for the High Speed Output to execute the command in the Command Register.
HSO__COMMAND	HSO Command Register - Determines what will happen at the time loaded into the HSO Time registers.
HSI__STATUS	HSI Status Registers - Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins. In Window 15 - Writes to pin detected bits, but not current state bits.
SBUF(TX)	Transmit buffer for the serial port, holds contents to be outputted. Last written value is readable in Window 15.
SBUF(RX)	Receive buffer for the serial port, holds the byte just received by the serial port. Writable in Window 15.
INT__MASK	Interrupt Mask Register - Enables or disables the individual interrupts.
INT__PEND	Interrupt Pending Register - Indicates that an interrupt signal has occurred on one of the sources and has not been serviced. (also INT__PENDING)
WATCHDOG	Watchdog Timer Register - Written periodically to hold off automatic reset every 64K state times. Returns upper byte of WDT counter in Window 15.
TIMER1	Timer 1 Hi/Lo - Timer1 high and low bytes.
TIMER2	Timer 2 Hi/Lo - Timer2 high and low bytes.
IOPORT0	Port 0 Register - Levels on pins of Port 0. Reserved in Window 15.
BAUD__RATE	Register which determines the baud rate, this register is loaded sequentially. Reserved in Window 15.
IOPORT1	Port 1 Register - Used to read or write to Port 1. Reserved in Window 15
IOPORT2	Port 2 Register - Used to read or write to Port 2.
SP__STAT	Serial Port Status - Indicates the status of the serial port.
SP__CON	Serial Port Control - Used to set the mode of the serial port.
IOS0	I/O Status Register 0 - Contains information on the HSO status. Writes to HSO pins in Window 15.
IOS1	I/O Status Register 1 - Contains information on the status of the timers and of the HSI.
IOC0	I/O Control Register 0 - Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.
IOC1	I/O Control Register 1 - Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.
PWM__CONTROL	Pulse Width Modulation Control Register - Sets the duration of the PWM pulse.
INT__PEND1	Interrupt Pending register for the 8 new interrupt vectors (also INT__PENDING1)
INT__MASK1	Interrupt Mask register for the 8 new interrupt vectors
IOC2	I/O Control Register 2 - Controls new 80C196KB features
IOS2	I/O Status Register 2 - Contains information on HSO events
WSR	Window Select Register - Selects register window

Figure 2-4. Special Function Register Description

Programming control and test operations are done in Window 14. Registers in this window that are not labeled should be considered reserved and should not be either read or written.

In register Window 15 (WSR = 15), the operation of the SFRs is changed, so that those which were read-only in Window 0 space are write-only and vice versa. The only major exception to this is that Timer2 is read/write in Window 0, and T2 Capture is read/write in Window 15. (Timer2 was read-only on the 8096.) Registers which can be read and written in Window 0 can also be read and written in Window 15.

Figure 2-4 contains brief descriptions of the SFR registers. Detailed descriptions are contained in the section which discusses the peripheral controlled by the register. Figure 2-5 contains a description of the alternate function in Window 15.

AD_COMMAND (02H)	— Read the last written command
AD_RESULT (02H, 03H)	— Write a value into the result register
HSI_MODE (03H)	— Read the value in HSI_MODE
HSI_TIME (04H, 05H)	— Write to FIFO Holding register
HSO_TIME (04H, 05H)	— Read the last value placed in the holding register
HSI_STATUS (06H)	— Write to status bits but not to HSI pin bits. (Pin bits are 1, 3, 5, 7)
HSO_COMMAND (06H)	— Read the last value placed in the holding register
SBUF(RX) (07H)	— Write a value into the receive buffer
SBUF(TX) (07H)	— Read the last value written to the transmit buffer
WATCHDOG (0AH)	— Read the value in the upper byte of the WDT
TIMER1 (0AH, 0BH)	— Write a value to Timer1
TIMER2 (0CH, 0DH)	— Read/Write the Timer2 capture register. (Timer2 read/write is done with WSR = 0)
IOC2 (0BH)	— Last written value is readable, except bit 7 (Note 1)
BAUD_RATE (0EH)	— No function, cannot be read
PORT0 (0EH)	— No function, no output drivers on the pins
SP_STAT (11H)	— Set the status bits, TI and RI can be set, but it will not cause an interrupt
SP_CON (11H)	— Read the current control byte
IOS0 (15H)	— Writing to this register controls the HSO pins. Bits 6 and 7 are inactive for writes.
IOC0 (15H)	— Last written value is readable, except bit 1 (Note 1)
IOS1 (16H)	— Writing to this register will set the status bits, but not cause interrupts. Bits 6 and 7 are not functional.
IOC1 (16H)	— Last written value is readable
IOS2 (17H)	— Writing to this register will set the status bits, but not cause interrupts.
PWM_CONTROL (17H)	— Read the duty cycle value written to PWM_CONTROL

NOTE:
1. IOC2.7 (CAM CLEAR) and IOC0.1 (T2RST) are not latched and will read as a 1 (precharged bus).

Being able to write to the read-only registers and vice-versa provides a lot of flexibility. One of the most useful advantages is the ability to set the timers and HSO lines for initial conditions other than zero.

Figure 2-5. Alternate SFR Function in Window 15

Within the SFR space are several registers and bit locations labeled "RESERVED". These locations should never be written or read. A reserved bit location should always be written with 0 to maintain compatibility with future parts. Values read from these locations may change from part to part or over temperature and voltage. Registers and bits which are not labeled should be treated as reserved registers and bits. Note that the default state of internal registers is 0, while that for external memory is 1. This is because SFR functions are typically disabled with a zero, while external memory is typically erased to all 1s.

Caution must be taken when using the SFRs as sources of operations or as base or index registers for indirect or indexed operations. It is possible to not get the desired results, since external events can change SFRs and some SFRs clear when read. The potential for an SFR to change value must be taken into account when operating on these registers. This is particularly important when high level languages are used as they do not always make allowances for SFR-type registers. SFRs can be operated on as bytes or words unless otherwise specified.

2.3 Reserved Memory Spaces

Locations 1FFEH and 1FFFH are used for Ports 3 and 4 respectively, allowing easy reconstruction of these ports if external memory is used. An example of reconstructing the I/O ports is given in Section 16. If ports 3 and 4 are not going to be reconstructed and internal ROM/EPROM is not used, these locations can be treated as any other external memory location.

Many reserved and special locations are in the memory area between 2000H and 2080H. In this area the 18 interrupt vectors, chip configuration byte, and security key are located. Figure 2-6 shows the locations and functions of these registers. The interrupts, chip configuration, and security key registers are discussed in Sections 5, 16, and 17 respectively. All unspecified addresses in locations 2000H through 207FH, including those marked "Reserved" are reserved by Intel for use in testing or future products. They must be filled with the Hex value FFH to insure compatibility with future parts.

	FFFFH
EXTERNAL MEMORY OR I/O	
	4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	
	2080H
RESERVED	2074H–207FH
VOLTAGE LEVELS	2072H–2073H
SIGNATURE WORD	2070H–2071H
RESERVED	2040H–206FH
INTERRUPT VECTORS	2030H–203FH
SECURITY KEY	2020H–202FH
RESERVED	201CH–201FH
SELF JUMP CODE (27H FEH)	201AH–201BH
RESERVED	2019H
CHIP CONFIGURATION BYTE	2018H
RESERVED	2015H–2017H
PPW	2014H
INTERRUPT VECTORS	2000H–2013H

Figure 2-6. Reserved Memory Spaces

Resetting the 80C196KB causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in Section 13.

2.4 Internal ROM and EPROM

When a ROM part is ordered, or an EPROM part is programmed, the internal memory locations 2080H through 3FFFH are user specified, as are the interrupt vectors, Chip Configuration Register and Security Key in locations 2000H through 207FH.

Instruction and data fetches from the internal ROM or EPROM occur only if the part has ROM or EPROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory. The $\overline{EA}$ pin is latched on $\overline{RESET}$ rising. Information on programming EPROMs can be found in Section 17.

The 80C196KB provides a ROM/EPROM lock feature to allow the program to be locked against reading and/or writing the internal program memory. In order to maintain security, code can not be executed out of the last three locations of internal ROM/EPROM if the lock is enabled. Details on this feature are in Section 17.

2.5 System Bus

There are several modes of system bus operation on the 80C196KB. The standard bus mode uses a 16-bit multiplexed address/data bus. Other bus modes include an 8-bit mode and a mode in which the bus size can dynamically be switched between 8-bits and 16-bits.

Hold/Hold Acknowledge ($\overline{HOLD}/\overline{HLDA}$) and Ready signals are available to create a variety of memory systems. The READY line extends the width of the $\overline{RD}$ (read) and $\overline{WR}$ (write) pulses to allow access of slow memories. The maximum number of wait states can be internally limited to 1, 2, or 3 by the value in the Chip Configuration Register (CCR). Multiple processor systems with shared memory can be designed using $\overline{HOLD}/\overline{HLDA}$ to keep the 80C196KB off the bus. Details on the System Bus are in Sections 15 and 16.

3.0 SOFTWARE OVERVIEW

This section provides information on writing programs to execute in the 80C196KB. Additional information can be found in the following documents:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048 (Intel Systems)
Order Number 122351 (DOS Systems)

MCS®-96 UTILITIES USER'S GUIDE
Order Number 122049 (Intel Systems)
Order Number 122356 (DOS Systems)

PL/M-96 USER'S GUIDE
Order Number 122134 (Intel Systems)
Order Number 122361 (DOS Systems)

C-96 USER'S GUIDE
Order Number 167632 (DOS Systems)

Throughout this chpater short sections of code are used to illustrate the operation of the device. For these sections it is assumed that the following set of temporary registers has been declared:

AX, BX, CX, and DX are 16-bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in the 8086. It is important to note that in the 80C196KB these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within the on-board register file.

3.1 Operand Types

The MCS-96 architecture supports a variety of data types likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language are used where appropriate. To avoid confusion, the name of an operand type is capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

BYTES

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7, with 0 being the least significant bit. There are no alignment restrictions for BYTES, so they may be placed anywhere in the MCS-96 address space.

WORDS

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

SHORT-INTEGERS

SHORT-INTEGERS are 8-bit signed variables which can take on the values between -128 and $+127$. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

INTEGERS

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and +32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

BITS

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 80C196KB provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

DOUBLE-WORDS

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 80196KB and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with Intel provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in Section 3.5.

LONG-INTEGERS

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and +2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 80C196KB and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in Section 3.5.

3.2 Operand Addressing

Operands are accessed within the address space of the 80C196KB with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic address modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

REGISTER-DIRECT REFERENCES

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and the register address must conform to the operand type's alignment rules. Depending on the instruction, up to three registers can take part in the calculation.

Examples

```
ADD    AX,BX,CX      ; AX:=BX+CX
MUL    AX,BX         ; AX:=AX*BX
INCB   CL            ; CL:=CL+1
```

INDIRECT REFERENCES

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 80C196KB, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

Examples

```
LD    AX,[AX]     ; AX:=MEM_WORD(AX)
ADDB  AL,BL,[CX]  ; AL:=BL+MEM_BYTE(CX)
POP   [AX]        ; MEM_WORD(AX):=MEM_WORD(SP) ; SP:=SP+2
```

INDIRECT WITH AUTO-INCREMENT REFERENCES

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

Examples

```
LD    AX,[BX]+    ; AX:=MEM_WORD(BX) ; BX:=BX+2
ADDB  AL,BL,[CX]+ ; AL:=BL+MEM_BYTE(CX) ; CX:=CX+1
PUSH  [AX]+       ; SP:=SP-2;
                  ;   MEM_WORD(SP):=MEM_WORD(AX)
                  ;   AX:=AX+2
```

IMMEDIATE REFERENCES

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

Examples

```
ADD  AX,#340  ; AX:=AX+340
PUSH #1234H   ; SP:=SP-2; MEM_WORD(SP):=1234H
DIVB AX,#10   ; AL:=AX/10; AH:=AX MOD 10
```

SHORT-INDEXED REFERENCES

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which contains an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended, the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
LD.   AX,12[BX]     ; AX:=MEM_WORD(BX+12)
MULB  AX,BL,3[CX]   ; AX:=BL*MEM_BYTE(CX+3)
```

LONG-INDEXED REFERENCES

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

```
Examples
 AND  AX,BX,TABLE[CX]     ; AX:=BX AND MEM_WORD(TABLE+CX)
 ST   AX,TABLE[BX]        ; MEM_WORD(TABLE+BX):=AX
 ADDB AL,BL,LOOKUP[CX]    ; AL:=BL+MEM_BYTE(LOOKUP+CX)
```

ZERO REGISTER ADDRESSING

The first two bytes in the register file are fixed at zero by the 80C196KB hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

```
Examples
 ADD  AX,1234[0]   ; AX:=AX+MEM_WORD(1234)
 POP  5678[0]      ; MEM_WORD(5678):=MEM_WORD(SP)
                   ;   SP:=SP+2
```

STACK POINTER REGISTER ADDRESSING

The system stack pointer in the 80C196KB is accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

```
Examples
 PUSH  [SP]       ; DUPLICATE TOP_OF_STACK
 LD    AX,2[SP]   ; AX:=NEXT_TO_TOP
```

ASSEMBLY LANGUAGE ADDRESSING MODES

The MCS-96 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Indexed Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

These features of the assembly language simplify the programming task and should be used wherever possible.

3.3 Program Status Word

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. There are two bytes in the PSW; the actual status word and the low byte of the interrupt mask. Figure 3-1 shows the status bits of the PSW. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF). Only the interrupt section of the PSW can be accessed directly. There is no SFR for the PSW status bits.

CONDITION FLAGS

The PSW bits on the 80C196KB are set as follows:

PSW:	7	6	5	4	3	2	1	0
	Z	N	V	VT	C	X	I	ST

Figure 3-1. PSW Register

Z: The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N: The Negative flag is set to indicate that the operation generated a negative result. Note that the N flag will be in the algebraically correct state even if an overflow occurs. For shift operations, including the normalize operation and all three forms (SHL, SHR, SHRA) of byte, word and double word shifts, the N flag will be set to the same value as the most significant bit of the result. This will be true even if the shift count is 0.

V: The oVerflow flag is set to indicate that the operation generated a result which is outside the range for the destination data type. For the SHL, SHLB and SHLL instructions, the V flag will be set if the most significant bit of the operand changes at any time during the shift. For divide operations, the following conditions are used to determine if the V flag is set:

For the operation:	V is set if Quotient is:
UNSIGNED BYTE DIVIDE	> 255(0FFH)
UNSIGNED WORD DIVIDE	> 65535(0FFFFH)
SIGNED BYTE DIVIDE	< −127(81H) or > 127(7FH)
SIGNED WORD DIVIDE	< −32767(8001H) or > 32767(7FFFH)

VT: The oVerflow Trap flag is set when the V flag is set, but it is only cleared by the CLRVT, JVT and JNVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C: The Carry flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation, or the state of the last bit shifted out of an operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C=0.)

X: Reserved for future. Should always be cleared when writing to the PSW for compatibility with future products.

I: The global Interrupt disable bit disables all interrupts except NMI when cleared.

ST: The ST (STicky bit) flag is set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL   ;AX:=CL*DL
SHR     AX,#4      ;Shift right 4
                    places
```

If the C flag is set after the shift, it indicates that the bits shifted off the end of the operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift, it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of the C flag alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C	ST	Value of the Bits Shifted Off
0	0	Value = 0
0	1	0 < Value < ½ LSB
1	0	Value = ½ LSB
1	1	Value > ½ LSB

Figure 3-2. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

INTERRUPT FLAGS

The lower eight bits of the PSW individually mask the lowest 8 sources of interrupt to the 80C196KB. These mask bits can be accessed as an eight bit byte (INT_MASK—address 8) in the on-board register file. A separate register (INT_MASK1—address 13H) contains the control bits for the higher 8 interrupts. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. Bit 9 in the PSW is the global interrupt disable. If this bit is cleared then interrupts will be locked out. Note that the interrupts are collected in the INT_PEND registers even if they are locked out. Execution of the corresponding service routines will proceed according to their priority when they become enabled. Further information on the interrupt structure of the 80C196KB can be found in Section 5.

3.4 Instruction Set

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of by 16 divides and for shift operations. The remaining operations on 32-bit variables can be implemented by combinations of 16-bit operations. As an example the sequence:

```
ADD     AX,CX
ADDC    BX,DX
```

performs a 32-bit addition, and the sequence

```
SUB     AX,CX
SUBC    BX,DX
```

performs a 32-bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 80C196KB (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 80C196KB NORML instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 80C196KB supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if unsigned operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Tables 3-1 and 3-2 summarize the operation of each of the instructions. Complete descriptions of each instruction and its timings can be found in the MCS-96 family Instruction Set chapter. Examples of using the instruction set of the MCS-96 family can be found in Chapter 5, "Using the 8096".

Table 3-1A. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	–	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	–	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	–	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	–	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	–	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	↕	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	↕	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	↕	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	↕	↑	–	
AND/ANDB	2	D ← D AND A	↕	↕	0	0	–	–	
AND/ANDB	3	D ← B AND A	↕	↕	0	0	–	–	
OR/ORB	2	D ← D OR A	↕	↕	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	↕	↕	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP − 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Table 3-1B. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC ← (SP); SP ← SP + 2	–	–	–	–	–	–	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	–	–	–	–	–	–	5
JC	1	Jump if C = 1	–	–	–	–	–	–	5
JNC	1	jump if C = 0	–	–	–	–	–	–	5
JE	1	jump if Z = 1	–	–	–	–	–	–	5
JNE	1	Jump if Z = 0	–	–	–	–	–	–	5
JGE	1	Jump if N = 0	–	–	–	–	–	–	5
JLT	1	Jump if N = 1	–	–	–	–	–	–	5
JGT	1	Jump if N = 0 and Z = 0	–	–	–	–	–	–	5
JLE	1	Jump if N = 1 or Z = 1	–	–	–	–	–	–	5
JH	1	Jump if C = 1 and Z = 0	–	–	–	–	–	–	5
JNH	1	Jump if C = 0 or Z = 1	–	–	–	–	–	–	5
JV	1	Jump if V = 0	–	–	–	–	–	–	5
JNV	1	Jump if V = 1	–	–	–	–	–	–	5
JVT	1	Jump if VT = 1; Clear VT	–	–	–	–	0	–	5
JNVT	1	Jump if VT = 0; Clear VT	–	–	–	–	0	–	5
JST	1	Jump if ST = 1	–	–	–	–	–	–	5
JNST	1	Jump if ST = 0	–	–	–	–	–	–	5
JBS	3	Jump if Specified Bit = 1	–	–	–	–	–	–	5,6
JBC	3	Jump if Specified Bit = 0	–	–	–	–	–	–	5,6
DJNZ/ DJNZW	1	D ← D − 1; If D ≠ 0 then PC ← PC + 8-bit offset	–	–	–	–	–	–	5 10
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	–	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	–	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	–	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	–	–	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↕	↕	0	0	–	–	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	–	–	
CLR/CLRB	1	D ← 0	1	0	0	0	–	–	
SHL/SHLB/SHLL	2	C ← msb - - - - - lsb ← 0	↕	↕	↕	↕	↑	–	7
SHR/SHRB/SHRL	2	0 → msb - - - - - lsb → C	↕	↕	↕	0	–	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb - - - - - lsb → C	↕	↕	↕	0	–	↕	7
SETC	0	C ← 1	–	–	1	–	–	–	
CLRC	0	C ← 0	–	–	0	–	–	–	

Table 3-1C. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	0	VT ← 0	–	–	–	–	0	–	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interupts (I ← 0)	–	–	–	–	–	–	
EI	0	Enable All Interupts (I ← 1)	–	–	–	–	–	–	
NOP	0	PC ← PC + 1	–	–	–	–	–	–	
SKIP	0	PC ← PC + 2	–	–	–	–	–	–	
NORML	2	Left shift till msb = 1; D ← shift count	↕	↕	0	–	–	–	7
TRAP	0	SP ← SP − 2; (SP) ← PC; PC ← (2010H)	–	–	–	–	–	–	9
PUSHA	1	SP ← SP-2; (SP) ← PSW; PSW ← 0000H; SP ← SP-2; (SP) ← IMASK1/WSR; IMASK1 ← 00H	0	0	0	0	0	0	
POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	↕	↕	↕	↕	↕	↕	
IDLPD	1	IDLE MODE IF KEY=1; POWERDOWN MODE IF KEY =2; CHIP RESET OTHERWISE	–	–	–	–	–	–	
CMPL	2	D-A	↕	↕	↕	↕	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

NOTES:
1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is done. Operands is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D,D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D,D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.
10. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

Table 3.2A. Instruction Execution State Times (1)

MNEMONIC	DIRECT	IMMED	INDIRECT		INDEXED	
			NORMAL*	A-INC*	SHORT*	LONG*
ADD (3-op)	5	6	7/10	8/11	7/10	8/11
SUB (3-op)	5	6	7/10	8/11	7/10	8/11
ADD (2-op)	4	5	6/8	7/9	6/8	7/9
SUB (2-op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3-op)	5	5	7/10	8/11	7/10	8/11
SUBB (3-op)	5	5	7/10	8/11	7/10	8/11
ADDB (2-op)	4	4	6/8	7/9	6/8	7/9
SUBB (2-op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
MUL (3-op)	16	17	18/21	19/22	19/22	20/23
MULU (3-op)	14	15	16/19	17/19	17/20	18/21
MUL (2-op)	16	17	18/21	19/22	19/22	20/23
MULU (2-op)	14	15	16/19	17/19	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3-op)	12	12	14/17	13/15	15/18	16/19
MULUB (3-op)	10	10	12/15	12/16	12/16	14/17
MULB (2-op)	12	12	14/17	15/18	15/18	16/19
MULUB (2-op)	10	10	12/15	13/15	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3-op)	5	6	7/10	8/11	7/10	8/11
AND (2-op)	4	5	6/8	7/9	6/8	7/9
OR (2-op)	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3-op)	5	5	7/10	8/11	7/10	8/11
ANDB (2-op)	4	4	6/8	7/9	6/8	7/9
ORB (2-op)	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD/LDB	4	5	5/8	6/8	6/9	7/10
ST/STB	4	5	5/8	6/9	6/9	7/10
LDBSE	4	4	5/8	6/8	6/9	7/10
LDBZE	4	4	5/8	6/8	6/9	7/10
BMOV	6+8 per word			6 + 11/14 per word		
PUSH (int stack)	6	7	9/12	10/13	10/13	11/14
POP (int stack)	8	–	10/12	11/13	11/13	12/14
PUSH (ext stack)	8	9	11/14	12/15	12/15	13/16
POP (ext stack)	11	–	13/15	14/16	14/16	15/17

*Times for (Internal/External) Operands

NOTE:
1. Execution times may be slightly higher depending on the instruction stream being executed. The 80C196KB has a four byte queue for prefetching instructions. The instruction execution times may be longer if the prefetch queue is empty. For sixteen bit mode, the prefetch queue is rarely empty, therefore, the minimum instruction times should apply.

Table 3.2B. Instruction Execution State Times

MNEMONIC		MNEMONIC	
PUSHF (int stack)	6	PUSHF (ext stack)	8
POPF (int stack)	7	POPF (ext stack)	10
PUSHA (int stack)	12	PUSHA (ext stack)	18
POPA (int stack)	12	POPA (ext stack)	18
TRAP (int stack)	16	TRAP (ext stack)	18
LCALL (int stack)	11	LCALL (ext stack)	13
SCALL (int stack)	11	SCALL (ext stack)	13
RET (int stack)	11	RET (ext stack)	14
CMPL	7	DEC/DECB	3
CLR/CLRB	3	EXT/EXTB	4
NOT/NOTB	3	INC/INCB	3
NEG/NEGB	3		
LJMP	7		
SJMP	7		
BR [indirect]	7		
JNST, JST	4/8 jump not taken/jump taken		
JNH, JH	4/8 jump not taken/jump taken		
JGT, JLE	4/8 jump not taken/jump taken		
JNC, JC	4/8 jump not taken/jump taken		
JNVT, JVT	4/8 jump not taken/jump taken		
JNV, JV	4/8 jump not taken/jump taken		
JGE, JLT	4/8 jump not taken/jump taken		
JNE, JE	4/8 jump not taken/jump taken		
JBC, JBS	5/9 jump not taken/jump taken		
DJNZ	5/9 jump not taken/jump taken		
DJNZW (Note 1)	5/9 jump not taken/jump taken		
NORML	8 + 1 per shift (9 for 0 shift)		
SHRL	7+ 1 per shift (8 for 0 shift)		
SHLL	7 + 1 per shift (8 for 0 shift)		
SHRAL	7 + 1 per shift (8 for 0 shift)		
SHR/SHRB	6 + 1 per shift (7 for 0 shift)		
SHL/SHLB	6 + 1 per shift (7 for 0 shift)		
SHRA/SHRAB	6 + 1 per shift (7 for 0 shift)		
CLRC	2		
SETC	2		
DI	2		
EI	2		
CLRVT	2		
NOP	2		
RST	15 (includes fetch of configuration byte)		
SKIP	3		
IDLPD	8/25 (proper key/improper key)		

NOTE:
1. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

3.5 80C196KB Instruction Set Additions and Differences

For users already familiar with the 8096BH, there are six instructions added to the standard MCS-96 instruction set to form the 80C196KB instruction set. All of the former instructions perform the same function, except as indicated in the next section. The new instructions and their descriptions are listed below:

PUSHA — PUSHes the PSW, INT_MASK, IMASK1, and WSR

POPA — POPs the PSW, INT_MASK, IMASK1, and WSR

IDLPD — Sets the part into IDLE or Powerdown mode

CMPL — Compare 2 long direct values

BMOV — Block move using 2 auto-incrementing pointers and a counter

DJNZW — Decrement Jump Not Zero using a Word counter (Not functional on current stepping.)

INSTRUCTION DIFFERENCES

Instruction times on the 80C196KB are shorter than those on the 8096 for many instructions. For example a 16×16 unsigned multiply has been reduced from 25 to 14 states. In addition, many zero and one operand instructions and most instructions using external data take one or two fewer state times.

Indexed and indirect operations relative to the stack pointer (SP) work differently on the 80C196KB than on the 8096. On the 8096, the address is calculated based on the un-updated version of the stack pointer. The 80C196KB uses the updated version. The offset for PUSH[SP], POP[SP], PUSH nn[SP] and POP nn[SP] instructions may need to be changed by a count of 2.

3.6 Software Standards and Conventions

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

REGISTER UTILIZATION

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively, some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

ADDRESSING 32-BIT OPERANDS

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

SUBROUTINE LINKAGE

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed onto the stack as two 16-bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE
(param1,param2,param3);
  DECLARE param1 BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

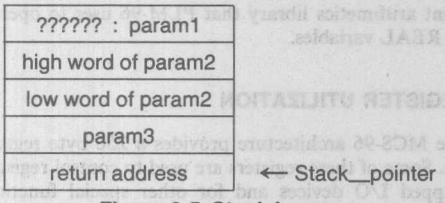


Figure 3-5. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8-, 16- or 32-bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a) Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b) Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c) The Program Status Word (PSW—see Section 3.3) is not saved and restored by procedures so the calling code must assume that the condition flags (Z, N, V, VT, C, and ST) are modified by the procedure.

d) Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

3.7 Software Protection Hints

Several features to assist in recovery from hardware and software errors are available on the 80C196KB. Protection is also provided against executing unimplemented opcodes by the unimplemented opcode interrupt. In addition, the hardware reset instruction (RST) can cause a reset if the program counter goes out of bounds. This instruction has an opcode of 0FFH, so if the processor reads in bus lines which have been pulled high it will reset itself.

It is recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed undesired results will occur. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 80C196KB instruction has 7 bytes) and a RST or jump to error routine instruction. Since RST is a one-byte instruction, the NOPs are not needed if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

The Watchdog Timer (WDT) further protects against software and hardware errors. When using the WDT to protect software it is desirable to reset it from only one place in code, lessening the chance of an undesired WDT reset. The section of code that resets the WDT should monitor the other code sections for proper operation. This can be done by checking variables to make sure they are within reasonable values. Simply using a software timer to reset the WDT every 10 milliseconds will provide protection only for catastrophic failures.

4.0 PERIPHERAL OVERVIEW

There are five major peripherals on the 80C196KB: the pulse-width-modulated output (PWM), Timer1 and Timer2, High Speed I/O Unit, Serial Port and A/O Converter. Minor peripherals on the 80C196KB include the watchdog timer and clock failure detect. With the exception of the high speed I/O unit (HSIO), each of the peripherals is a single unit that can be discussed without further separation.

Four individual sections make up the HSIO and work together to form a very flexible timer/counter based I/O system. Included in the HSIO are a 16-bit timer (Timer1), a 16-bit up/down counter (Timer2), a programmable high speed input unit (HSI), and a programmable high speed output unit (HSO). With very little CPU overhead the HSIO can measure pulse widths, generate waveforms, and create periodic interrupts. Depending on the application, it can perform the work of up to 18 timer/counters and capture/compare registers.

A brief description of the peripheral functions and interractions is included in this section. It provides overview information prior to the detailed discussions in the following sections. All of the details on control bits and precautions are in the individual sections for each peripheral starting with Section 5.

4.1 Pulse Width Modulation Output (D/A)

Digital to analog conversion can be done with the Pulse Width Modulation output. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 4-1.

The output waveform is a variable duty cycle pulse which repeats every 256 state times or 512 state times if the prescaler is enabled. Changes in the duty cycle are made by writing to the PWM register. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

4.2 Timer1 and Timer2

Two 16-bit timers are available for use on the 80C196KB. The first is designated "Timer1", the second "Timer2". Timer1 is used to synchronize events to real time, while Timer2 is clocked externally and synchronizes events to external occurrences. The timers are the time bases for the High Speed Input (HSI) and High Speed Output (HSO) units and can be considered an integral part of the HSI/O. A block diagram of the Timers is shown in Figure 4-2.

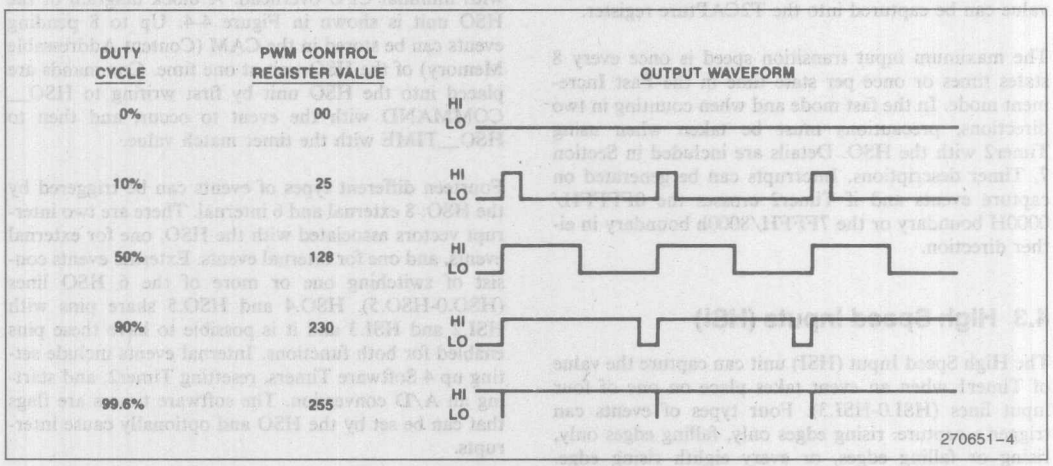


Figure 4-1. Typical PWM Outputs

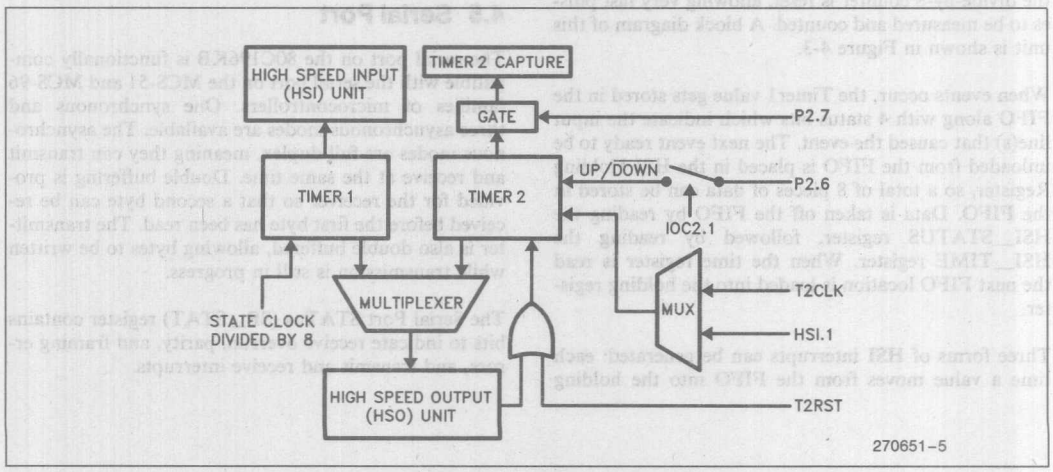


Figure 4-2. Timer Block Diagram

Timer1

Timer1 is a free-running timer which is incremented every eight state times, just as it is on the 8096. It can be read and written, but care must be taken when writing to it if the High Speed I/O (HSIO) Subsystem is being used. The precautions necessary when writing to Timer1 are described in Section 7. Timer1 can cause an interrupt when it overflows.

Timer2

Timer2 counts transitions, both positive and negative, on its input which can be either the T2CLK pin or the HSI.1 pin. Timer2 can be read and written and can be reset by hardware, software or the HSO unit. It can be used as an up/down counter based on Port 2.6 and it's value can be captured into the T2CAPture register.

The maximum input transition speed is once every 8 states times or once per state time in the Fast Increment mode. In the fast mode and when counting in two directions, precautions must be taken when using Timer2 with the HSO. Details are included in Section 7, Timer descriptions. Interrupts can be generated on capture events and if Timer2 crosses the 0FFFFH/0000H boundary or the 7FFFH/8000h boundary in either direction.

4.3 High Speed Inputs (HSI)

The High Speed Input (HSI) unit can capture the value of Timer1 when an event takes place on one of four input lines (HSI.0-HSI.3). Four types of events can trigger a capture: rising edges only, falling edges only, rising or falling edges, or every eighth rising edge. Whenever the every eighth rising edge mode is entered the divide-by-8 counter is reset, allowing very fast pulses to be measured and counted. A block diagram of this unit is shown in Figure 4-3.

When events occur, the Timer1 value gets stored in the FIFO along with 4 status bits which indicate the input line(s) that caused the event. The next event ready to be unloaded from the FIFO is placed in the HSI Holding Register, so a total of 8 pieces of data can be stored in the FIFO. Data is taken off the FIFO by reading the HSI_STATUS register, followed by reading the HSI_TIME register. When the time register is read the next FIFO location is loaded into the holding register.

Three forms of HSI interrupts can be generated: each time a value moves from the FIFO into the holding register; when the FIFO (independent of the holding register) has 4 or more events stored; and when the FIFO has 6 or more events stored. This flexibility allows optimization of the HSI for the expected frequency of interrupts.

Independent of the HSI operation, the state of the HSI lines is indicated by 4 bits of the HSI_STATUS register. Also independent of the HSI operation is the HSI.0 pin interrupt, which can be used as an extra external interrupt even if the pin is not enabled to the HSI unit.

4.4 High Speed Outputs (HSO)

The High Speed Output (HSO) unit can generate events at specified times or counts based on Timer1 or Timer2 with minimal CPU overhead. A block diagram of the HSO unit is shown in Figure 4-4. Up to 8 pending events can be stored in the CAM (Content Addressable Memory) of the HSO unit at one time. Commands are placed into the HSO unit by first writing to HSO_COMMAND with the event to occur, and then to HSO_TIME with the timer match value.

Fourteen different types of events can be triggered by the HSO: 8 external and 6 internal. There are two interrupt vectors associated with the HSO, one for external events, and one for internal events. External events consist of switching one or more of the 6 HSO lines (HSO.0-HSO.5). HSO.4 and HSO.5 share pins with HSI.2 and HSI.3 and it is possible to have these pins enabled for both functions. Internal events include setting up 4 Software Timers, resetting Timer2, and starting an A/D conversion. The software timers are flags that can be set by the HSO and optionally cause interrupts.

4.5 Serial Port

The serial port on the 80C196KB is functionally compatible with the serial port on the MCS-51 and MCS-96 families of microcontrollers. One synchronous and three asynchronous modes are available. The asynchronous modes are full duplex, meaning they can transmit and receive at the same time. Double buffering is provided for the receiver so that a second byte can be received before the first byte has been read. The transmitter is also double buffered, allowing bytes to be written while transmission is still in progress.

The Serial Port STATus (SP_STAT) register contains bits to indicate receive overrun, parity, and framing errors, and transmit and receive interrupts.

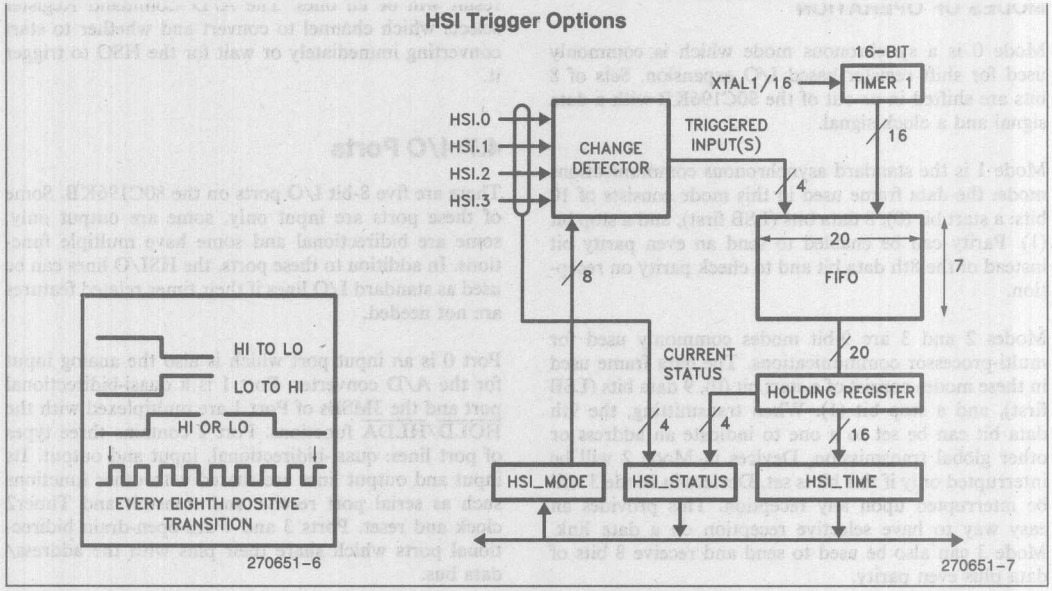


Figure 4-3. HSI Block Diagram

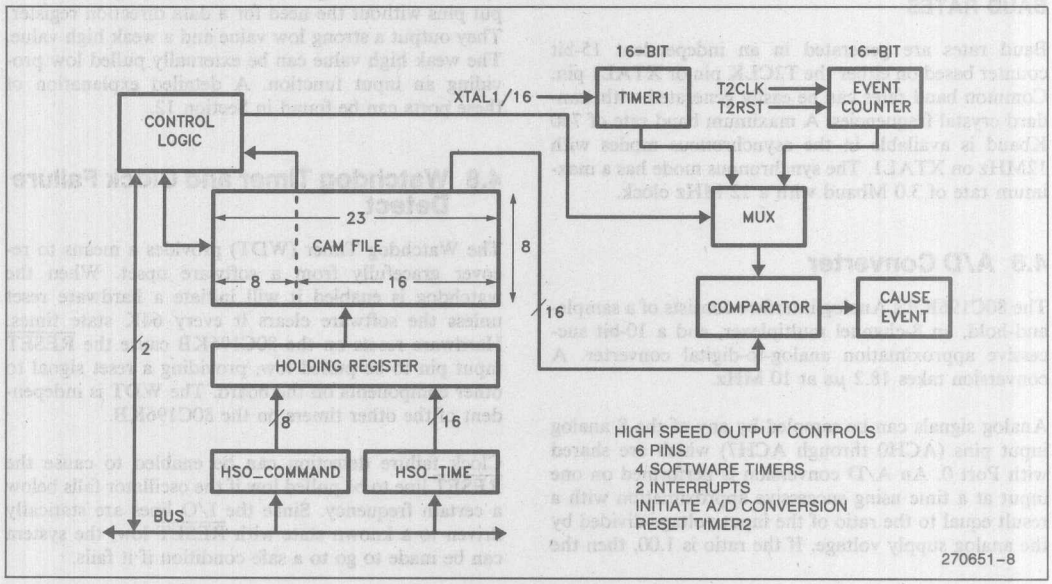


Figure 4-4. HSO Block Diagram

MODES OF OPERATION

Mode 0 is a synchronous mode which is commonly used for shift register based I/O expansion. Sets of 8 bits are shifted in or out of the 80C196KB with a data signal and a clock signal.

Mode 1 is the standard asynchronous communications mode: the data frame used in this mode consists of 10 bits: a start bit (0), 8 data bits (LSB first), and a stop bit (1). Parity can be enabled to send an even parity bit instead of the 8th data bit and to check parity on reception.

Modes 2 and 3 are 9-bit modes commonly used for multi-processor communications. The data frame used in these modes consist of a start bit (0), 9 data bits (LSB first), and a stop bit (1). When transmitting, the 9th data bit can be set to a one to indicate an address or other global transmission. Devices in Mode 2 will be interrupted only if this bit is set. Devices in Mode 3 will be interrupted upon any reception. This provides an easy way to have selective reception on a data link. Mode 3 can also be used to send and receive 8 bits of data plus even parity.

BAUD RATES

Baud rates are generated in an independent 15-bit counter based on either the T2CLK pin or XTAL1 pin. Common baud rates can be easily generated with standard crystal frequencies. A maximum baud rate of 750 Kbaud is available in the asynchronous modes with 12MHz on XTAL1. The synchronous mode has a maximum rate of 3.0 Mbaud with a 12 MHz clock.

4.6 A/D Converter

The 80C196KB's Analog interface consists of a sample-and-hold, an 8-channel multiplexer, and a 10-bit successive approximation analog-to-digital converter. A conversion takes 18.2 μs at 10 MHz.

Analog signals can be sampled by any of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. An A/D conversion is performed on one input at a time using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D Command Register selects which channel to convert and whether to start converting immediately or wait for the HSO to trigger it.

4.7 I/O Ports

There are five 8-bit I/O ports on the 80C196KB. Some of these ports are input only, some are output only, some are bidirectional and some have multiple functions. In addition to these ports, the HSI/O lines can be used as standard I/O lines if their timer related features are not needed.

Port 0 is an input port which is also the analog input for the A/D converter. Port 1 is a quasi-bidirectional port and the 3MSBs of Port 1 are multiplexed with the HOLD/HLDA functions. Port 2 contains three types of port lines: quasi-bidirectional, input and output. Its input and output lines are shared with other functions such as serial port receive and transmit and Timer2 clock and reset. Ports 3 and 4 are open-drain bidirectional ports which share their pins with the address/data bus.

Quasi-bidirectional pins can be used as input and output pins without the need for a data direction register. They output a strong low value and a weak high value. The weak high value can be externally pulled low providing an input function. A detailed explanation of these ports can be found in Section 12.

4.8 Watchdog Timer and Clock Failure Detect

The Watchdog Timer (WDT) provides a means to recover gracefully from a software upset. When the watchdog is enabled it will initiate a hardware reset unless the software clears it every 64K state times. Hardware resets on the 80C196KB cause the RESET input pin to be pulled low, providing a reset signal to other components on the board. The WDT is independent of the other timers on the 80C196KB.

Clock failure detection can be enabled to cause the RESET line to be pulled low if the oscillator falls below a certain frequency. Since the I/O lines are statically driven to a known state with RESET low, the system can be made to go to a safe condition if it fails.

5.0 INTERRUPTS

Twenty-eight (28) sources of interrupts are available on the 80C196KB. These sources are gathered into 15 vectors plus special vectors for NMI, the TRAP instruction, and Unimplemented Opcodes. Figure 5-1 shows the routing of the interrupt sources into their vectors as well as the control bits which enable some of the sources.

Special Interrupts

Three special interrupts are available on the 80C196KB: NMI, TRAP and Unimplemented opcode. The external NMI pin generates an unmaskable interrupt for implementation of critical interrupt routines. The TRAP instruction is useful in the development of custom software debuggers or generation of software interrupts. The unimplemented opcode interrupt generates an interrupt when unimplemented opcodes are exe-

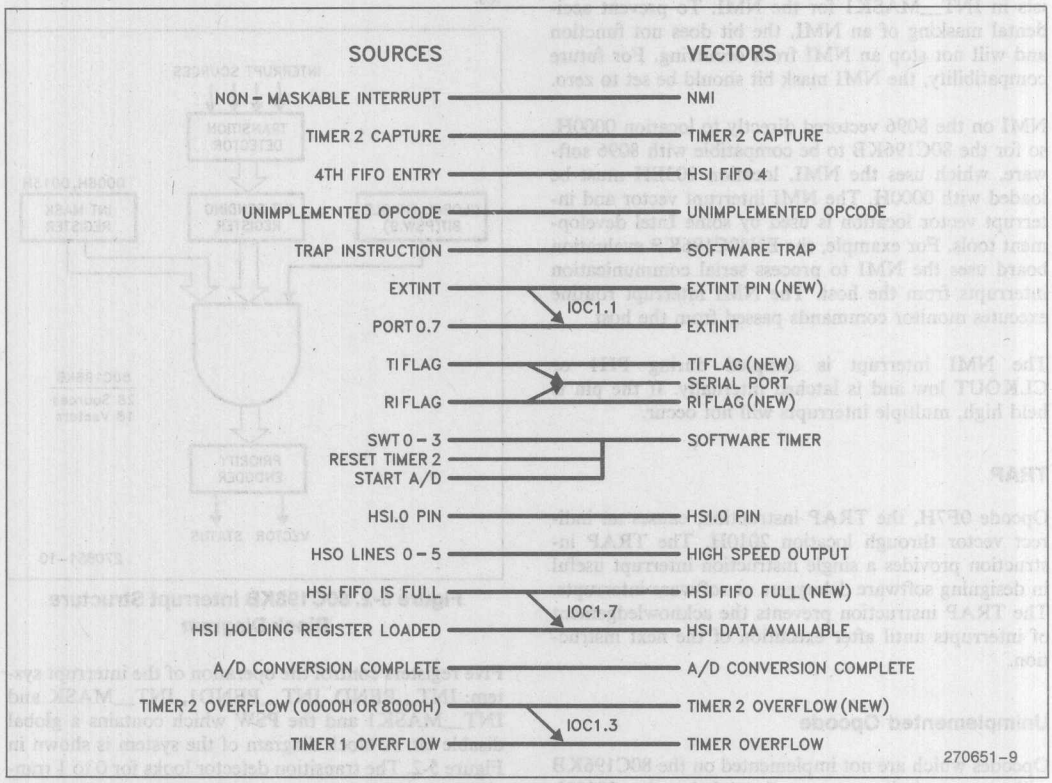


Figure 5-1. 80C196KB Interrupt Sources

cuted. This provides software recovery from random execution during hardware and software failures. Although available for customer use, these interrupts may be used in Intel development tools or evaluation boards.

NMI

NMI, the external Non-Maskable Interrupt, is the highest priority interrupt. It vectors indirectly through location 203EH. For design symmetry, a mask bit exists in INT_MASK1 for the NMI. To prevent accidental masking of an NMI, the bit does not function and will not stop an NMI from occurring. For future compatibility, the NMI mask bit should be set to zero.

NMI on the 8096 vectored directly to location 0000H, so for the 80C196KB to be compatible with 8096 software, which uses the NMI, location 203EH must be loaded with 0000H. The NMI interrupt vector and interrupt vector location is used by some Intel development tools. For example, the EV80C196KB evaluation board uses the NMI to process serial communication interrupts from the host. The NMI interrupt routine executes monitor commands passed from the host.

The NMI interrupt is sampled during PH1 or CLKOUT low and is latched internally. If the pin is held high, multiple interrupts will not occur.

TRAP

Opcode 0F7H, the TRAP instruction, causes an indirect vector through location 2010H. The TRAP instruction provides a single instruction interrupt useful in designing software debuggers or software interrupts. The TRAP instruction prevents the acknowledgement of interrupts until after execution of the next instruction.

Unimplemented Opcode

Opcodes which are not implemented on the 80C196KB will cause an indirect vector through location 2012H. User code or hardware which may have failed and run into an unimplemented opcode can software recover through this interrupt. The DJNZW instruction is not supported on the 80C196KB but remains a valid opcode, therefore, no interrupt will occur.

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routines. It is suggested that any unused interrupts be vectored to an error handling routine. The error routine should contain recovery code that will not further corrupt an already erroneous situation. In a debug environment, it may be desirable to have the routine lock into a jump to self loop which would be easily traceable with emulation tools. More sophisticated routines may be appropriate for production code recoveries.

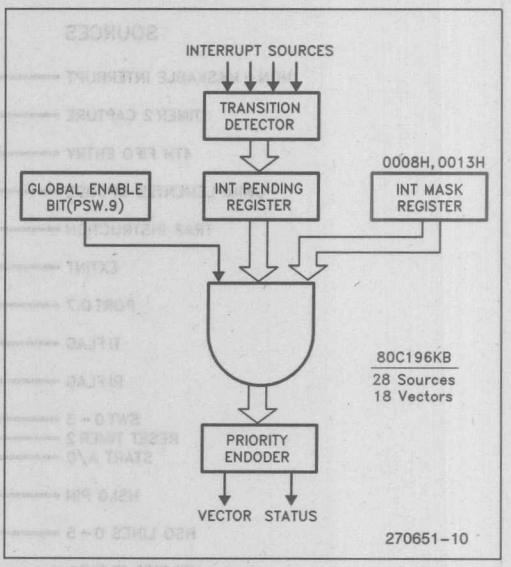


Figure 5-2. 80C196KB Interrupt Structure Block Diagram

Five registers control the operation of the interrupt system: INT_PEND, INT_PEND1, INT_MASK and INT_MASK1 and the PSW which contains a global disable bit. A block diagram of the system is shown in Figure 5-2. The transition detector looks for 0 to 1 transitions on any of the sources. External sources have a maximum transition speed of one edge every state time. Sampling will be guaranteed if the level on the interrupt line is held for at least one state time. If the interrupt line is not held for at least one state time, the interrupt may not be detected.

5.1 Interrupt Control

Interrupt Pending Register

When the hardware detects one of the sixteen interrupts it sets the corresponding bit in one of two pending interrupt registers (INT_PEND-09H and INT_PEND1-12H). When the interrupt vector is taken, the pending bit is cleared. These registers, the formats of which are shown in Figure 5-3, can be read or modified as byte registers. They can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PEND registers should ensure that the entire operation is inseparable. The easiest way to do this is to use the logical instructions in the two or three operand format, for example:

```
ANDB  INT_PEND,#11111101B
          ; Clears the A/D Interrupt
ORB   INT_PEND,#00000010B
          ; Sets the A/D Interrupt
```

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 5 state time "partial" interrupt cycle will occur. This is because the 80C196KB will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra two NOPs. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 80C196KB holds off acknowledging interrupts during these "read/modify/write" instructions.

POPF POP Flags pops the PSW/IMASK pair off the stack

Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask registers (INT_MASK-08H and INT_MASK1-13H). The format of these registers is the same as that of the Interrupt Pending Register shown in Figure 5-3.

The INT_MASK and INT_MASK1 registers can be read or written as byte registers. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The hardware will save any interrupts that occur by setting bits in the pending register, even if the interrupt mask bit is cleared. The INT_MASK register is the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags. Both the INT_MASK and INT_MASK1 registers can be saved with the PUSHA and POPA Instructions.

Global Disable

The processing of all interrupts except the NMI, TRAP and unimplemented opcode interrupts can be disabled by clearing the I bit in the PSW. Setting the I bit will enable interrupts that have mask register bits which are set. The I bit is controlled by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts. Interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

5.2 Interrupt Priorities

The priority encoder looks at all of the interrupts which are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Figure 5-4 (15 is highest, 0 is lowest). The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

		7	6	5	4	3	2	1	0
12H	IPEND1:	NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI
11H	IMASK1:								

		7	6	5	4	3	2	1	0
09H	IPEND:	EXT INT	SER PORT	SOFT TIMER	HSI.0 PIN	HSO PIN	HSI DATA	A/D DONE	TIMER OVF
08H	IMASK:								

Figure 5-3. Interrupt Mask and Pending Registers

Number	Source	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT Pin	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

Figure 5-4. 80C196KB Interrupt Priorities

This priority selection controls the order in which pending interrupts are passed to the software via interrupt calls. The software can then implement its own priority structure by controlling the mask registers (INT_MASK and INT_MASK1). To see how this is done, consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
 PUSHF          ; Save the PSW
                  (Includes INT_MASK)
 LDB    INT_MASK,#00000100B
 EI             ; Enable interrupts again
 ;  )
 ;  |
 ;  |
 ;  }   Service the interrupt
 ;  |
 ;  |
 ;  )
 POPF           ; Restore the PSW
 RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 80C196KB interrupt service routine execute properly:

a) After the hardware decides to process an interrupt, it generates and executes a special interrupt-call instruction, which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts, the first instruction of the interrupt service routine will execute.

b) The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global disable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and, by the time the LD instruction starts, all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c) The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d) The EI instruction reenables the processing of interrupts.

e) The actual interrupt service routine executes within the priority structure established by the software.

f) At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT_MASK register (part of the PSW), so any

changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

5.3 Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PEND
ANDB    AL,#bit_mask
STB     AL,INT_PEND
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PEND register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PEND contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT_PEND will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PEND to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSI interrupt The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 80C196KB allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction.

```
ANDB    INT_PEND,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. Changes to the INT_PEND register must be made as a single instruction, since bits can be changed in this register even if interrupts are disabled. Depending on system configurations, several other SFRs might also need to be changed in a single instruction for the same reason.

When variables must be modified without interruption, and a single instruction can not be used, the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

5.4 Interrupt Timing

The 80C196KB can be interrupted from four different external sources; NMI, EXTINT, HSI.0 and P0.7. All external interrupts are sampled during PH1 or CLKOUT low and are latched internally. Holding levels on external interrupts for at least one state time will ensure recognition of the interrupts.

The external interrupts on the 80C196KB, although sampled during PH1, are edge triggered interrupts as opposed to level triggered. Edge triggered interrupts will generate only one interrupt if the input is held high. On the other hand, level triggered interrupts will generate multiple interrupts when held high.

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt may not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI — Enable and disable all interrupts by toggling the global disable bit (PSW.9).

PUSHF — PUSH Flags pushes the PSW/INT_MASK pair then clears it, leaving both INT_MASK and PSW.9 clear.

POPF — POP Flags pops the PSW/INT_MASK pair off the stack

PUSHA — PUSH All does a PUSHF, then pushes the INT_MASK1/WSR pair and clears INT_MASK1

POPA — POP All pops the INT_MASK1/WSR pair and then does a POPF

Interrupts can also not occur immediately after execution of:

Unimplemented Opcodes

TRAP — The software trap instruction

SIGND — The signed prefix for multiply and divide instructions

When an interrupt is acknowledged the interrupt pending bit is cleared, and a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 16 state times. If the stack is in external RAM an additional 2 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 80C196KB begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 39 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (16(internal stack) or 18(external stack) state times). Therefore, the maximum response time is 61 (39 + 4 + 18) state times. This does not include the 10 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled. Refer to Figure 5-5, Interrupt Response Time, to visualize an example of worst case scenario.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The DI, PUSHF, POPF, PUSHA, POPA and TRAP instructions will also cause the same situation. Typically these instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

5.5 Interrupt Summary

Many of the interrupt vectors on the 8096 were shared with other interrupts. The interrupts which were shared on the 8096 are: Transmit Interrupt, Receive Interrupt, HSI FIFO Full, Timer2 Overflow and EXTINT. On the 80C196KB, these interrupts have their own interrupt vectors. The source of the interrupt vectors are typically programmed through control registers. These registers can be read in Window 15 to determine the source of any interrupt. Interrupt sources with two possible interrupt vectors, serial receive interrupt sharing serial port and receive interrupt vectors for example, should be configured for only one interrupt vector.

Interrupts with separate vectors include: NMI, TRAP, Unimplemented Opcode, Timer2 Capture, 4th Entry into HSI FIFO, Software timer, HSI.0 Pin, High Speed Outputs, and A/D conversion Complete. The NMI, TRAP and Unimplemented Opcode interrupts were covered in section 5.1.

EXTINT and P0.7

The external interrupt pin can vector through either EXTINT Pin interrupt vector (203AH) or External Interrupt vector (200AH). To be 8096 compatible, IOC1.1 controls the source for the external interrupt vector to be either EXTINT or P0.7. On the 80C196KB, the EXTINT pin interrupt vector is independent of the external interrupt vector shared with Port0.7, therefore, P0.7 and EXTINT can be configured as separate interrupts. The EXTINT pin should not be programmed to vector through both interrupt vector locations.

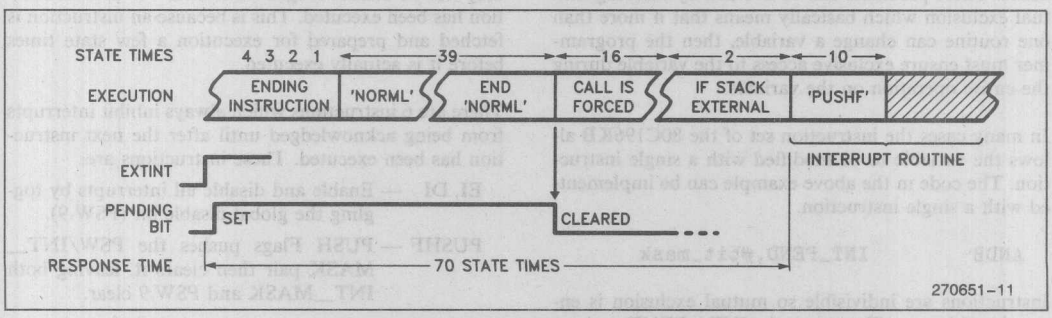


Figure 5-5. Interrupt Response Time

Serial Port Interrupts

The serial port generates one of three possible interrupts: Transmit interrupt TI(2030H), Receive Interrupt RI(2032H) and SERIAL(200CH). Refer to section 10 for information on the serial port interrupts. The 8096 shared the TI and RI interrupts on the SERIAL interrupt vector. On the 80C196KB, these interrupts share both the serial interrupt vector and have their own interrupt vectors. Ideally, the transmit and receive interrupts should be programmed as separate interrupt vectors while disabling the SERIAL interrupt. For 8096 compatibility, the interrupts can still use the SERIAL interrupt vector.

HSI FIFO FULL and HSI DATA AVAILABLE

HSI FIFO FULL and HSI DATA AVAILABLE interrupts shared the HSI DATA AVAILABLE interrupt vector on the 8096. The source of the HSI DATA AVAILABLE interrupt is controlled by the setting of I/O Control Register 1,(IOC1.7). Setting IOC1.7 to zero will generate an interrupt when a time value is loaded into the holding register. Setting the bit to one generates an interrupt when the FIFO, independent of the holding register, has six entries in it.

On the 80C196KB, separate interrupt vectors are available for the HSI FIFO FULL(203CH) and HSI DATA AVAILABLE(2004H) interrupts. The interrupts should be programmed for separate interrupt vector locations. Refer to Section 8 for more information on the High Speed Inputs.

HSI FIFO_4

The HSI FIFO can generate an interrupt when the HSI has four or more entries in the FIFO. The HSI FIFO_4 interrupt vectors through location 2034H. Refer to Section 8 for more information on the High Speed Inputs.

HSI.0 External Interrupt

The rising edge on HSI.0 pin can be used as an external interrupt. The HSI.0 pin is sampled during PH1 or CLKOUT low. The setup time for sampling is 30 nSEC before the falling edge of CLKOUT. Sampling is guaranteed if the pin is held for at least one state time. The interrupt vectors through location 2008H. The pin must first be enabled by setting IOC0.0 before an interrupt can occur.

Timer2 and Timer1 overflow

Timer2 and Timer1 can interrupt on overflow. These interrupts shared the same interrupt vector TIMER OVERFLOW(2000H) on the 8096. The source of the interrupt vector is controlled by I/O Control Register 1 bit 3. If the bit is set, Timer2 overflow will be the source, otherwise, it will be Timer1. On the 80C196KB Timer2 overflow(0H or 8000h) has a separate interrupt vector through location 2038H.

Timer2 Capture

The 80C196KB can generate an interrupt in response to a Timer2 capture triggered by a rising edge on P2.7. Timer2 Capture vectors through location 2036H.

High Speed Outputs

The High Speed Outputs interrupt can be generated in response to a programmed HSO command which causes an external event. HSO commands which set or clear the High Speed Output pins are considered external events. Status Register IOS2 indicates which HSO events have occured and can be used to arbitrate which HSO command caused the interrupt. The High Speed Output interrupt vectors indirectly through location 2006H. For more information on High Speed Outputs, refer to Section 9.

Software Timers

HSO commands which create internal events can interrupt through the Software Timer interrupt vector. Internal events include triggering an A/D conversion, resetting Timer2 and software timers. Status registers IOS2 and IOS1 can be used to determine which internal HSO event has occured. Location 200AH is the interrupt vector for the Software Timer interrupt. Refer to Section 9 for more information on software timers and the HSO.

A/D Conversion Complete

The A/D Conversion Complete interrupt can generate an interrupt in response to a completed A/D conversion. The interrupt vectors indirectly through location 2002H. Refer to section 11 for more information on the A/D Converter.

6.0 Pulse Width Modulation Output (D/A)

Digital to analog conversion can be done with the Pulse Width Modulation output; a block diagram of the circuit is shown in Figure 6-1. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in

Figure 6-2. Note that when the PWM register equals 00, the output is always low. Additionally, the PWM register will only be reloaded from the temporary latch when the counter overflows. This means the compare circuit will not recognize a new value until the counter has expired preventing missed PWM edges.

The 80C196KB PWM unit has a prescaler bit (divide by 2) which is enabled by setting IOC2.2 = 1. The PWM frequencies are shown in Figure 6-3. The output waveform is a variable duty cycle pulse which repeats every 256 or 512 state times (42.75 μs or 85.5 μs at 12 MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. The value programmed into the PWM register can be read in Window 15 (WSR = 15). There are several types of motors which require a PWM waveform for more efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle. as described in the next section.

XTAL1 =	8 MHz	10 MHz	12 MHz
IOC2.2 = 0	15.6 KHz	19.6 KHz	23.6 KHz
IOC2.2 = 1	7.8 KHz	9.8 KHz	11.8 KHz

Figure 6-3. PWM Frequencies

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function.

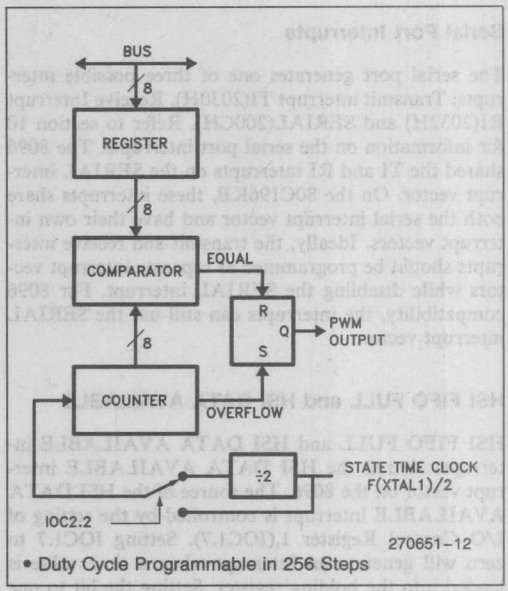


Figure 6-1. PWM Block Diagram

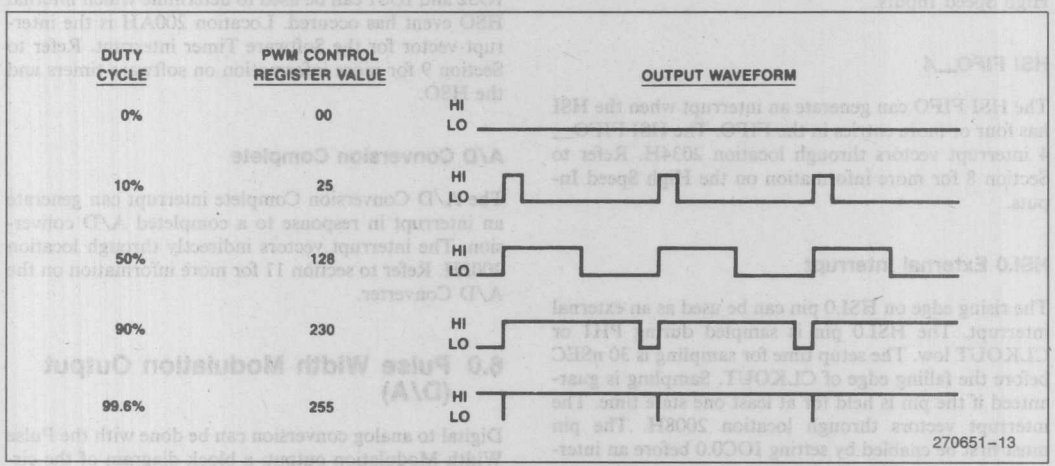


Figure 6-2. Typical PWM Outputs

6.1 Analog Outputs

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. See Section 9.7 for information on generating a PWM with the High Speed Output Unit. Either device will generate a rectangular pulse train that varies in duty cycle and period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to low current levels. A block diagram of the type of circuit needed is shown in Figure 6-4. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. Figure 6-5 shows two typical circuits. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can generate these waveforms if a fixed period on the order of 64 μs is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 87.5 milliseconds. Both of these outputs produce CHMOS levels.

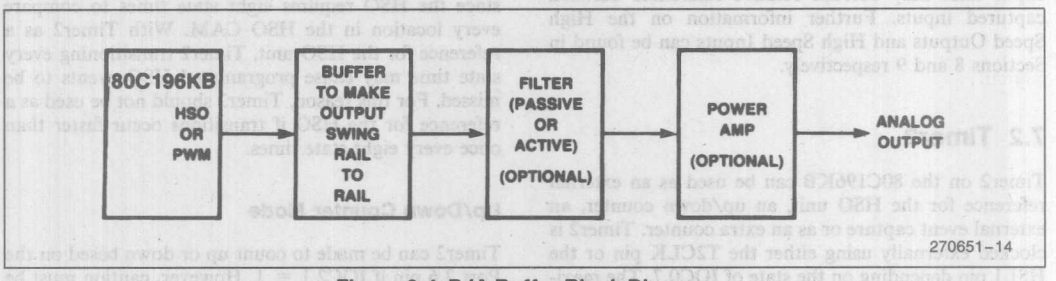


Figure 6-4. D/A Buffer Block Diagram

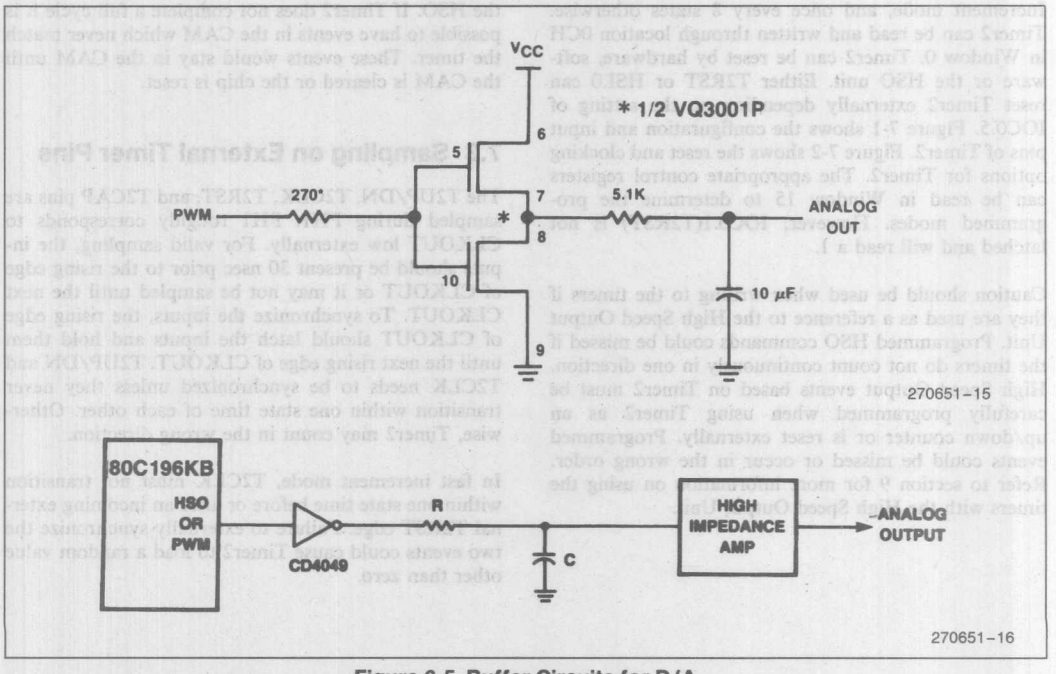


Figure 6-5. Buffer Circuits for D/A

7.0 TIMERS

7.1 Timer1

Timer1 is a 16-bit free-running timer which is incremented every eight state times. An interrupt can be generated in response to an overflow. It is read through location 0AH in Window 0 and written in Window 15. Care must be taken when writing to it if the High Speed I/O (HSIO) Subsystem is being used. HSO time entries in the CAM depend on exact matches with Timer1. Writes to Timer1 should be taken into account in software to ensure events in the HSO CAM are not missed or occur in an order which may be unexpected. Changing Timer1 with incoming events on the High Speed Input lines may corrupt relative references between captured inputs. Further information on the High Speed Outputs and High Speed Inputs can be found in Sections 8 and 9 respectively.

7.2 Timer2

Timer2 on the 80C196KB can be used as an external reference for the HSO unit, an up/down counter, an external event capture or as an extra counter. Timer2 is clocked externally using either the T2CLK pin or the HSI.1 pin depending on the state of IOC0.7. The maximum transition speed is once per state time in the Fast Increment mode, and once every 8 states otherwise. Timer2 can be read and written through location 0CH in Window 0. Timer2 can be reset by hardware, software or the HSO unit. Either T2RST or HSI.0 can reset Timer2 externally depending on the setting of IOC0.5. Figure 7-1 shows the configuration and input pins of Timer2. Figure 7-2 shows the reset and clocking options for Timer2. The appropriate control registers can be read in Window 15 to determine the programmed modes. However, IOC0.1(T2RST) is not latched and will read a 1.

Caution should be used when writing to the timers if they are used as a reference to the High Speed Output Unit. Programmed HSO commands could be missed if the timers do not count continuously in one direction. High Speed Output events based on Timer2 must be carefully programmed when using Timer2 as an up/down counter or is reset externally. Programmed events could be missed or occur in the wrong order. Refer to section 9 for more information on using the timers with the High Speed Output Unit.

Capture Register

The value in Timer2 can be captured into the T2CAPture register by a rising edge on P2.7. The edge must be held for at least one state time as discussed in the next section. T2CAP is located at 0CH in Window 15. The interrupt generated by a capture vectors through location 2036H.

Fast Increment Mode

Timer2 can be programmed to run in fast increment mode to count transitions every state time. Setting IOC2.0 programs Timer2 in the Fast Increment mode. In this mode, the events programmed on the HSO unit with Timer2 as a reference will not execute properly since the HSO requires eight state times to compare every location in the HSO CAM. With Timer2 as a reference for the HSO unit, Timer2 transitioning every state time may cause programmed HSO events to be missed. For this reason, Timer2 should not be used as a reference for the HSO if transitions occur faster than once every eight state times.

Up/Down Counter Mode

Timer2 can be made to count up or down based on the Port 2.6 pin if IOC2.1 = 1. However, caution must be used when this feature is working in conjunction with the HSO. If Timer2 does not complete a full cycle it is possible to have events in the CAM which never match the timer. These events would stay in the CAM until the CAM is cleared or the chip is reset.

7.3 Sampling on External Timer Pins

The T2UP/DN, T2CLK, T2RST, and T2CAP pins are sampled during PH1. PH1 roughly corresponds to CLKOUT low externally. For valid sampling, the inputs should be present 30 nsec prior to the rising edge of CLKOUT or it may not be sampled until the next CLKOUT. To synchronize the inputs, the rising edge of CLKOUT should latch the inputs and hold them until the next rising edge of CLKOUT. T2UP/DN and T2CLK needs to be synchronized unless they never transition within one state time of each other. Otherwise, Timer2 may count in the wrong direction.

In fast increment mode, T2CLK must not transition within one state time before or after an incoming external T2RST edge. Failure to externally synchronize the two events could cause Timer2 to load a random value other than zero.

	Bit = 1	Bit = 0
IOC0.1	Reset Timer2 each write	No action
IOC0.3	Enable external reset	Disable
IOC0.5	HSI.0 is ext. reset source	T2RST is reset source
IOC0.7	HSI.1 is T2 clock source	T2CLK is clock source
IOC1.3	Enable Timer2 overflow int.	Disable overflow interrupt
IOC2.0	Enable fast increment	Disable fast increment
IOC2.1	Enable downcount feature	Disable downcount
P2.6	Count down if IOC2.1 = 1	Count up
IOC2.5	Interrupt on 7FFFH/8000H	Interrupt on 0FFFFH/0000H
P2.7	Capture Timer2 into T2CAPture on rising edge	

Figure 7-1. Timer2 Configuration and Control Pins

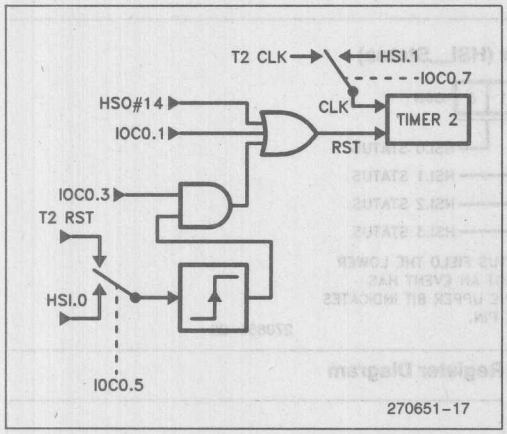


Figure 7-2. Timer2 Clock and Reset Options

7.4 Timer Interrupts

Both Timer1 and Timer2 can trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). Timer1 overflow is controlled by setting IOC1.2 and the interrupt status is indicated in IOS1.5. The TIMER OVERFLOW interrupt is enabled by setting INT_MASK.0.

A Timer2 overflow condition interrupts through location 2000H by setting IOC1.3 and setting INT_MASK.0. Alternatively, Timer2 overflow can interrupt through location 2038H by setting INT_MASK1.3. The status of the Timer2 overflow interrupt is indicated in IOS1.4.

Interrupts can be generated if Timer2 crosses the 0FFFFH/0000H boundary or the 7FFFH/8000H boundary in either direction. By having two interrupt points it is possible to have interrupts enabled even if Timer2 is counting up and down centered around one of the interrupt points. The boundaries used to control the Timer2 interrupt is determined by the setting of IOC2.5. When set, Timer2 will interrupt on the 7FFFH/8000H boundary, otherwise, the 0FFFFH/0000H boundary interrupts.

A T2CAPTURE interrupt is enabled by setting INT_MASK1.3. The interrupt will vector through location 2036H.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears bits 0 through 5 including the software timer flags. It is, therefore, recommended to copy the byte to a temporary register before testing bits. Writing to IOS1 in Window 15 will set the status bits but not cause interrupts. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

8.0 HIGH SPEED INPUTS

The High Speed Input Unit (HSI) can record the time an event occurs with respect to Timer1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 are bidirectional pins which can also be used as HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) determine the functions of these pins. The values programmed into IOC0 and IOC1 can be read in Window 15. A block diagram of the HSI unit is shown in Figure 8-1.

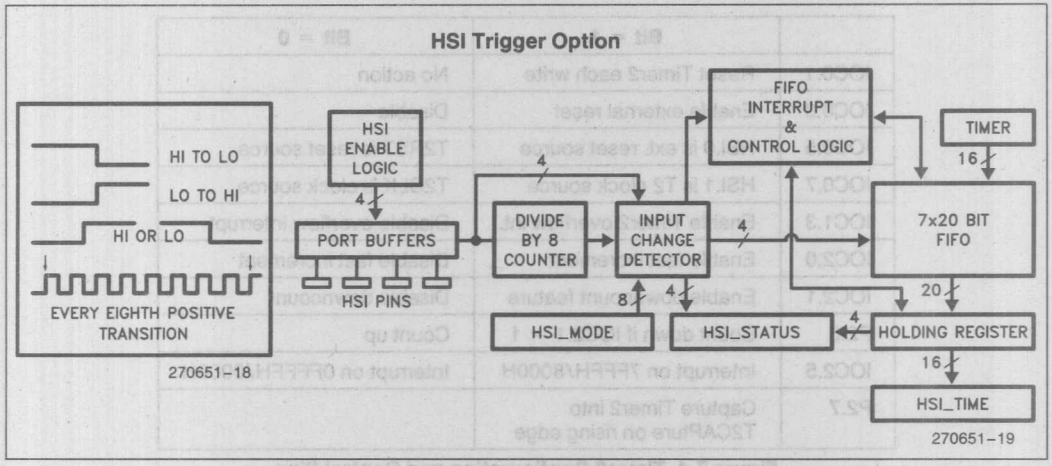


Figure 8-1. High Speed Input Unit

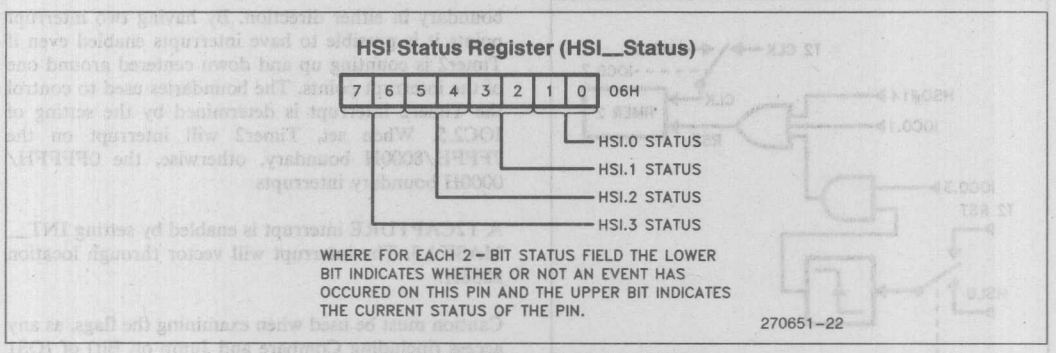


Figure 8-2. HSI Status Register Diagram

When an HSI event occurs, a 7×20 FIFO stores the 16 bits of Timer1, and the 4 bits indicating which pins had events. It can take up to 8 state times for this information to reach the holding register. For this reason, 8 state times must be allowed between consecutive reads of HSI_TIME. When the FIFO is full, one additional event, for a total of 8 events, can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full, any additional events will not be recorded.

8.1 HSI Modes

There are 4 possible modes of operation for each of the HSI pins. The HSI_MODE register at location 03H controls which pins will look for what type of events. In Window 15, reading the register will read back the programmed HSI mode. The 8-bit register is set up as shown in Figure 8-3.

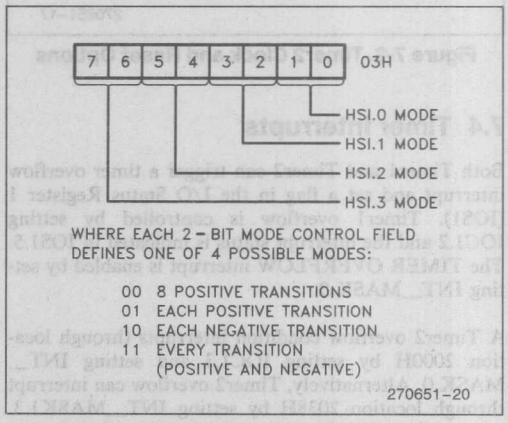


Figure 8-3. HSI Mode Register 1

High and low levels each need to be held for at least 1 state time to ensure proper operation. The HSI inputs are sampled during PH1 or CLKOUT low and require 30 nsec of setup to the rising edge of CLKOUT. Edges which missed the setup time will not be recognized until the next CLKOUT. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 8-4 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO. The status of the HSI pins can be read in the HSI_STATUS register at location 06H, shown in Figure 8-2.

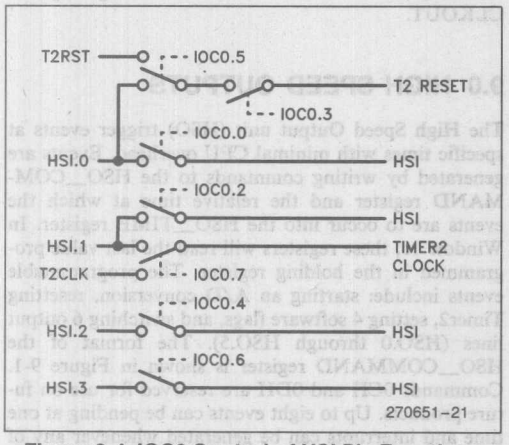


Figure 8-4. IOC0 Control of HSI Pin Functions

8.2 HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. The register IOS1 is displayed in Figure 8-5. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the HSI holding register has data available to be read. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears bits 0–5, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. Bits 6 and 7 are not functional in Window 15.

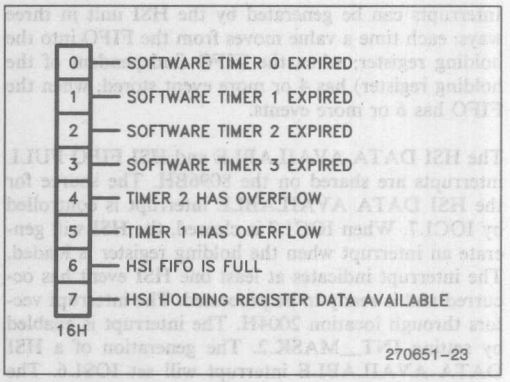


Figure 8-5. I/O Status Register 1

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI_STATUS Register is shown in Figure 8-2. Second, the HSI Time register is read. Reading the Time register unloads one level of the FIFO, so if the Time register is read before the Status register, the event information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI_TIME register is read without the holding register being loaded, the returned value will be indeterminate. Under the same conditions, the four bits in HSI_STATUS indicating which events have occurred will also be indeterminate. The four HSI_STATUS bits which indicate the current state of the pins will always return the correct value.

It should be noted that many of the Status register conditions are changed by a reset, see section 13. Writing

to HSI_TIME in window 15 will write to the HSI FIFO holding register. Writing to HSI_STATUS in Window 15 will set the status bits but will not affect the HSI pins.

8.3 HSI Interrupts

Interrupts can be generated by the HSI unit in three ways: each time a value moves from the FIFO into the holding register; when the FIFO (independent of the holding register) has 4 or more event stored; when the FIFO has 6 or more events.

The HSI DATA AVAILABLE and HSI FIFO FULL interrupts are shared on the 8096BH. The source for the HSI DATA AVAILABLE interrupt is controlled by IOC1.7. When IOC1.7 is cleared, the HSI will generate an interrupt when the holding register is loaded. The interrupt indicates at least one HSI event has occurred and is ready to be processed. The interrupt vectors through location 2004H. The interrupt is enabled by setting INT_MASK.2. The generation of a HSI DATA AVAILABLE interrupt will set IOS1.6. The HSI FIFO FULL interrupt will vector through HSI DATA AVAILABLE if IOC1.7 is set. On the 80C196KB, the HSI FIFO FULL has a separate interrupt vector at location 203CH.

A HSI FIFO FULL interrupt occurs when the HSI FIFO has six or more entries loaded independent of the holding register. Since all interrupts are rising edge triggered, the processor will not be reinterrupted until the FIFO first contains 5 or less records, then contains six or more. The HSI FIFO FULL interrupt mask bit is INT_MASK1.6. The occurrence of a HSI FIFO FULL interrupt is indicated by setting IOS1.7. Earlier warning of a impending FIFO full condition can be achieved by the HSI FIFO 4th Entry interrupt.

The HSI_FIFO_4 interrupt generates an interrupt when four or more events are stored in the HSI FIFO independent of the holding register. The interrupt is enabled by setting INT_MASK1.2. The HSI_FIFO_4 vectors indirectly through location 2034H. There is no status flag associated with the HSI_FIFO_4 interrupt since it has its own independent interrupt vector.

The HSI.0 pin can generate an interrupt on the rising edge even if its not enabled to the HSI FIFO. An interrupt generated by this pin vectors through location 2008H.

8.4 HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. The actual sampling occurs during PH1 or during CLKOUT low. The HSI inputs should be valid at least 30 nsec before the rising of CLKOUT. Otherwise, the HSI input may be sampled in the next CLKOUT. Therefore, if information is to be synchronized to the HSI it should be latched on the rising edge of CLKOUT.

9.0 HIGH SPEED OUTPUTS

The High Speed Output unit (HSO) trigger events at specific times with minimal CPU overhead. Events are generated by writing commands to the HSO_COMMAND register and the relative time at which the events are to occur into the HSO_TIME register. In Window 15, these registers will read the last value programmed in the holding register. The programmable events include: starting an A/D conversion, resetting Timer2, setting 4 software flags, and switching 6 output lines (HSO.0 through HSO.5). The format of the HSO_COMMAND register is shown in Figure 9-1. Commands 0CH and 0DH are reserved for use on future products. Up to eight events can be pending at one time and interrupts can be generated whenever any of these events are triggered. HSO.4 and HSO.5 are bidirectional pins which are multiplexed with HSI.2 and HSI.3 respectively. Bits 4 and 6 of I/O Control Register 1 (IOC1.4, IOC1.6) enable HSO.4 and HSO.5 as outputs. The Control Registers can be read in Window 15 to determine the programmed modes for the HSO. However, the IOC2.7(CAM CLEAR) bit is not latched and will read as a one. Entries can be locked in the CAM to generate periodic events or waveforms.

	7	6	5	4	3	2	1	0	
HSO_COMMAND	CAM LOCK	TMR2/$\overline{\text{TMR1}}$	SET/$\overline{\text{CLEAR}}$	INT/$\overline{\text{INT}}$	CHANNEL				06H

CAM Lock — Locks event in CAM if this is enabled by IOC2.6 (ENA_LOCK)

TMR/$\overline{\text{TMR1}}$ — Events Based on Timer2/Based on Timer1 if 0

SET/$\overline{\text{CLEAR}}$ — Set HSO line/Clear HSO line if 0

INT/$\overline{\text{INT}}$ — Cause interrupt/No interrupt if 0

CHANNEL (in Hex):
- 0–5: HSO lines 0–5
- 6: HSO lines 0 and 1
- 7: HSO lines 2 and 3
- 8–B: Software Timers 0–4
- C–D: Unflagged Events (Do not use for future compatibility)
- E: Reset Timer2
- F: Start A to D Conversion

Figure 9-1. HSO Command Register

9.1 HSO Interrupts and Software Timers

The HSO unit can generate two types of interrupts. The High Speed Output execution interrupt can be generated (if enabled) for HSO commands which change one or more of the six output pins. The other HSO interrupt is the interrupt which can be generated by any other HSO command, (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

HSO Interrupt Status

Register IOS2 at location 17H displays the HSO events which have occurred. IOS2 is shown in Figure 9-2. The events displayed are HSO.0 through HSO.5, Timer2 Reset and start of an A/D conversion. IOS2 is cleared when accessed, therefore, the register should be saved in an image register if more than one bit is being tested. The status register is useful in determining which events have caused an HSO generated interrupt. Writing to this register in Window 15 will set the status bits but not cause interrupts. In Window 15, writing to IOS2 can set the High Speed Output lines to an initial value. Refer to Section 15.2 for more information on Window 15.

IOS2:	7	6	5	4	3	2	1	0
	START A/D	T2 RESET	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

17H read — Indicates which HSO event occcured

START A/D: HSO_CMD 15, start A/D

T2RESET: HSO_CMD 14, Timer2 Reset

HSO.0–5: Output pins HSO.0 through HSO.5

Figure 9-2. I/O Status Register 2

SOFTWARE TIMERS

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the HSO command register was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer2 or starts an A/D conversion, it can also be programmed to generate a software timer interrupt.

If more than one software timer interrupt occurs in the same time frame, multiple status bits will be set. Each read or test of any bit in IOS1 (see Figure 9-5) will clear bits 0 through 5. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 11.5.

9.2 HSO CAM

A block diagram of the HSO unit is shown in Figure 9-3. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with the timer values every state time, taking 8 state times to compare all CAM registers with the timers. This defines the time resolution of the HSO to be 8 state times (1.33 microseconds at an oscillator frequency of 12 MHz).

Each CAM register is 24 bits wide. Sixteen bits specify the time at which the action is to be carried out, one bit for the lock bit and 7 bits specify both the nature of the action and whether Timer1 or Timer2 is the reference. The format of the command to the HSO unit is shown in Figure 9-1. Note that bit 5 is ignored for command channels 8 through 0FH.

To enter a command into the CAM file, write the 8-bit "Command Tag" into location 0006H followed by the time the action is to be carried out into word address 0004H. The typical code would be:

```
LDB HSO_COMMAND,#what_to_do
ADD HSO_TIME,Timer1,#when_to_do_it
```

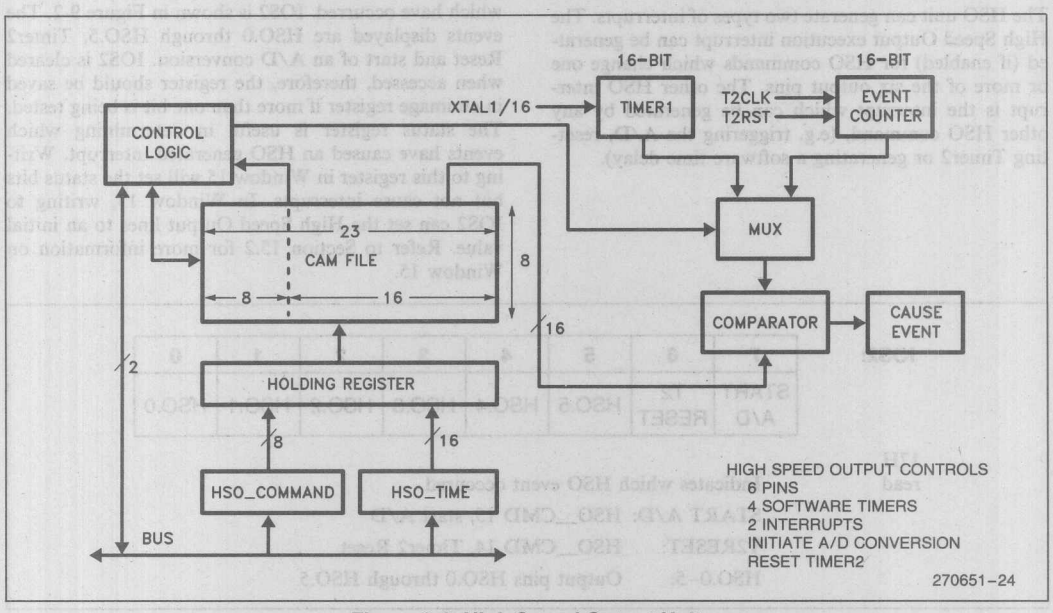


Figure 9-3. High Speed Output Unit

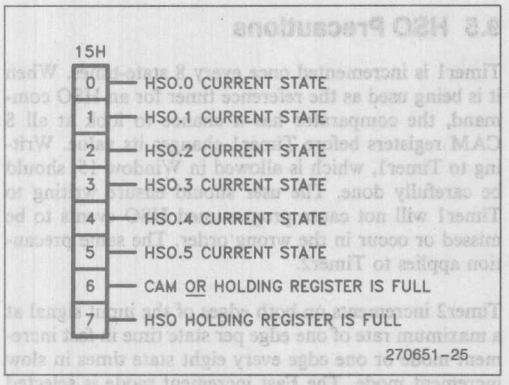


Figure 9-4. I/O Status Register 0

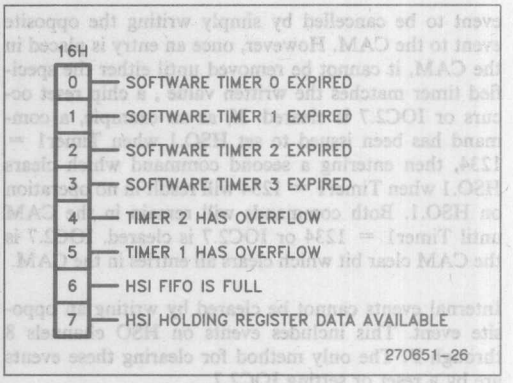


Figure 9-5. I/O Status Register 1 (IOS1)

Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM to execute. Commands in the holding register can also be overwritten. Since it can take up to 8 state times for a command to move from the holding register to the CAM, 8 states must be allowed between successive writes to the CAM.

To provide proper synchronization, the minimum time that should be loaded to Timer1 is Timer1 + 2. Smaller values may cause the Timer match to occur 65,636 counts later than expected. A similar restriction applies if Timer2 is used.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit.

9.3 HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty. The programmer should carefully decide which of these two flags is the best to use for each application. This register also shows the current status of the HSO.0 through HSO.5. The HSO pins can be set by writing to this register in Window 15. The format for I/O Status Register 0 is shown in Figure 9-4.

The expiration of software timer 0 through 4, and the overflow of Timer1 and Timer2 are indicated in IOS1. The status bits can be set in Window 15 but not cause interrupts. The register is shown in Figure 9-5.

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LDB   AL,IOS1
```

but also to implicit reads such as:

```
JB    IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which maintains an image of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB   IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on, then the modification of IOS1_image must be done from inside a critical region.

9.4 Clearing the HSO and Locked Entries

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external

event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value , a chip reset occurs or IOC2.7 is cleared. If, as an example, a command has been issued to set HSO.1 when Timer1 = 1234, then entering a second command which clears HSO.1 when Timer1 = 1234 will result in no operation on HSO.1. Both commands will remain in the CAM until Timer1 = 1234 or IOC2.7 is cleared. IOC2.7 is the CAM clear bit which clears all entries in the CAM.

Internal events cannot be cleared by writing an opposite event. This includes events on HSO channels 8 through F. The only method for clearing these events are by a reset or setting IOC2.7.

HSO LOCKED ENTRIES

The CAM Lock bit (HSO_Command.7) can be set to keep commands in the CAM, otherwise the commands will clear from the CAM as soon as they cause an event. This feature allows for generation periodic events based on Timer2 and must be enabled by setting IOC2.6. To clear locked events from the CAM, the entire CAM can be cleared by writing a one to the CAM clear bit IOC2.7. A chip reset will also clear the CAM.

Locked entries are useful in applications requiring periodic or repetitive events to occur. Timer2 used as an HSO reference can generate periodic events with the use of the HSO T2RST command. HSO events programmed with a HSO time less then the Timer2 reset time will occur repeatedly as Timer2 resets. Recurrent software tasks can be scheduled by locking software timers commands into the High Speed Output Unit. Interrupt service routines can be written to execute the tasks without having to program another HSO software timer command. Continuous sampling of the A/D converter can be accompished by programming a locked HSO A/D conversion command. The GO bit must still be set in the A/D interrupt routine but the HSO command does not have be reloaded. One of the most useful features is the generation of multiple PWM's on the High Speed Output lines. Locked entries provide the ability to program periodic events while minimizing the software overhead. Section 9.7 describes the generation of four PWMs using locked entries.

Individual external events setting or clearing an HSO pin can by cancelled by writing the opposite event to the CAM. The HSO events do not occur until the timer reference has changed state. An event programmed to set and clear an HSO event at the same time will cancel each other out. Locked entries can correspondingly be cancelled using this method. However, the entries remain in the HSO CAM and can quickly fill up the available eight locations. As an alternative, all entries in the HSO CAM can be cleared by setting IOC2.7.

9.5 HSO Precautions

Timer1 is incremented once every 8 state-times. When it is being used as the reference timer for an HSO command, the comparator has a chance to look at all 8 CAM registers before Timer1 changes its value. Writing to Timer1, which is allowed in Window 15, should be carefully done. The user should ensure writing to Timer1 will not cause programmed HSO events to be missed or occur in the wrong order. The same precaution applies to Timer2.

Timer2 increments on both edges of the input signal at a maximum rate of one edge per state time in fast increment mode or one edge every eight state times in slow increment mode. The Fast increment mode is selected by setting IOC2.1 high. The HSO requires at least eight state times to compare each entry in the CAM. Therefore, the fast increment mode for Timer2 cannot be used as a reference for the HSO if transitions occur faster then once every eight state times.

Referencing events when Timer2 is being used as an up/down counter could cause events to occur in opposite order or be missed entirely. Additionally, locked entries could possibly occur several times if Timer2 is oscillating around the time tag for an entry. For these reasons, it is recommended that Timer2 count in only one direction when used as a reference for the HSO.

When using Timer2 as the HSO reference, caution must be taken that Timer2 is not reset prior to the highest value for a Timer2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs. If that match is to a time value on Timer2 which is never reached, the event will remain pending in the CAM until the part is reset or CAM is cleared.

A potential problem occurs when resetting Timer2 with an external reset pin. This situation arises when the event is set to occur when Timer2 is equal to zero. If HSI.0 or the T2RST pin clears Timer2, and Timer2 equal to zero triggers the event, then the event may not occur. This is because HSI.0 and T2RST clear Timer2 asynchronously, and Timer2 may then be incremented to one before the HSO CAM entry can be compared and acted upon. This can be avoided by setting the event to occur when Timer2 is equal to one. This method will ensure that there is enough time for the CAM entry to be recognized.

9.6 PWM Using the HSO

The HSO unit can generate PWM waveforms with very little CPU overhead using Timer2 as a reference. A PWM is generated by programming an HSO line to a

high and a T2RST to occur at the same time. An HSO low time is programmed on the CAM to generate the duty cycle of the PWM. A repetitive PWM waveform is generated by locking the commands into the CAM. Reprogramming of the duty cycle or PWM frequency can be accomplished by generating a software interrupt and reprogramming the HSO high, HSO low and T2RST commands.

Multiple PWMs can be programmed using Timer2 as a reference and locked CAM entries. Up to four PWM's can be generated by locking a PWM(High) and PWM(low) into the CAM for each HSO.0 through HSO.3. Timer2 is used as a reference and set to zero by programming a T2RST command at the same time an HSO command sets all the lines high. Two CAM entries program the four PWM (high) times by setting HSO.0/HSO.1 and HSO.2/HSO.3 high with the same command. Four entries in the CAM set each of the HSO lines low. One entry is used to reset Timer2. This method uses a total of seven CAM entries with little or no software overhead. The PWMs can change their duty cycle by reprogramming the CAM with different HSO levels.

Changing the duty cycle for each PWM requires the flushing of the CAM and reprogramming of all seven entries in the CAM. The 80C196KB can flush the entire CAM by setting bit 7 in the IOC2 register (location 16H). Each HSO(high) and HSO(low) times should be reprogrammed in addition to the Timer2 reset command. This method provides for up to four PWM's with no software overhead except when reprogramming the duty cycle of any particular PWM. The code to generate these PWMs is shown in Figure 9-6.

```
$include (regl96.inc)
; ********************************************************
; *                                                      *
; *     GENERATION OF FOUR PWM'S USING LOCKED ENTRIES    *
; *                                                      *
; *     Timer2 is used as a reference and is clocked     *
; *     externally by T2CLK.  The High Speed outputs are *
; *     used as PWMs by programming each individual      *
; *     PWM(low) and PWM(High) time as a locked entry.   *
; *     The period of the PWM is programmed by resetting *
; *     timer2 and setting all the HSO lines high at the *
; *     same time.  The PWMs are reprogrammed by         *
; *     clearing the HSO CAM and reloading new values    *
; *     for the PWM period and duty cycle.               *
; *                                                      *
; ********************************************************

RSEG at 60h
pwm0timl: dsw 1
pwm1timl: dsw 1
pwm2timl: dsw 1
pwm3timl: dsw 1
PWM_period: dsw 1
temp: dsw 1

cseg at 2080h
     ld sp,#0d0h              ; initialize stack pointer
     ld PWM_period,#0f000h    ; intialize pwm period
     ld pwm0timl,#2000h       ; initialize pwm 0-3 duty cycle
     ld pwm1timl,#4000h
     ld pwm2timl,#6000h
     ld pwm3timl,#8000h
     ldb ioc2,#40h            ; Enable locked entries
     ldb ioc0,#0h             ; Enable t2clk for timer2 clock
                              ; source
     call pwm_program         ; program pwm's on CAM
here:     sjmp here           ; loop forever
```

Figure 9-6. Generating Four PWMs Using Locked Entries

```
pwm_program:
      ldb ioc2,#0c0h      ; flush entire cam
      ldb hso_command,#0ceh ; program timer2 reset time
      ld hso_time,PWM_period
      nop                 ; delay eight state times before
      nop                 ; next load
      nop
      nop
      ldb hso_command,#0e6h  ; HSO 0/1 high, locked, timer2 as
                          ; reference
      ld hso_time,PWM_period ; set hso_high on t2rst
      nop
      nop
      nop
      nop
      ldb hso_command,#0e7h  ; HSO 1/2 high, locked, timer2
                          ; as reference
      ld hso_time,PWM_period ; set hso_high on t2rst
      nop
      nop
      nop
      nop
      ldb hso_command,#0c0h  ; set HSO.0 low, locked, timer2
                          ; as reference
      ld hso_time,pwm0timl   ; HSO.0 time low
      nop
      nop
      nop
      nop
      ldb hso_command,#0c1h  ; set HSO.1 low, locked, timer2
                          ; reference
      ld hso_time,pwm1timl   ; HSO.1 time low
      nop
      nop
      nop
      nop
      ldb hso_command,#0c2h  ; set HSO.2 low, locked,timer2
                          ; as reference
      ld hso_time,pwm2timl   ; HSO.2 time low
      nop
      nop
      nop
      nop
      ldb hso_command,#0c3h  ; set HSO.3 low, locked,timer2
                          ; as reference
      ld hso_time,pwm3timld  ; HSO.3 time low
      ret
      end
```

Figure 9-6. Generating Four PWMs Using Locked Entries (Continued)

9.7 HSO Output Timing

Changes in the HSO lines are synchronized to either Timer1 or Timer2. All of the external HSO lines due to change at a certain value of a timer will change just after the incrementing of the timer. Internally, the timer changes every eight state times during Phase1. From an external perspective the HSO pin should change just prior to the falling edge of CLKOUT and be stable by its rising edge. Information from the HSO can be latched on the CLKOUT rising edge. Internal events also occur when the reference timer increments.

10.0 SERIAL PORT

The serial port on the 80C196KB has one synchronous and 3 asynchronous modes. The asynchronous modes are full duplex, meaning they can transmit and receive at the same time. The receiver is double buffered so that the reception of a second byte can begin before the first byte has been read. The transmitter on the 80C196KB is also double buffered allowing continuous transmissions. The port is functionally compatible with the serial port on the MCS-51 family of microcontrollers, although the software controlling the ports is different.

Data to and from the serial port is transferred through SBUF(RX) and SBUF(TX), both located at 07H. SBUF(TX) holds data ready for transmission and SBUF(RX) contains data received by the serial port. SBUF(TX) and SBUF(RX) can be read and can be written in Window 15.

Mode 0, the synchronous shift register mode, is designed to expand I/O over a serial line. Mode 1 is the standard 8 bit data asynchronous mode used for normal serial communications. Modes 2 and 3 are 9 bit data asynchronous modes typically used for interprocessor communications. Mode 2 provides monitoring of a communication line for a 1 in the 9th bit position before causing an interrupt. Mode 3 causes interrupts independant of the 9th bit value.

10.1 Serial Port Status and Control

Control of the serial port is done through the Serial Port Control (SP_CON) register shown in Figure 10-1. Writing to location 11H accesses SP_CON while reading it accesses SP_STAT. Note that reads of SP_STAT will return indeterminate data in the lower 2 bits and writing to the upper 3 bits of SP_CON has no effect on chip functionality and must be written as OS for future compatibility. On the 80C196KB the SP_STAT register contains new bits to indicate receive Overrun Error (OE), Framing Error (FE), and Transmitter Empty (TXE). The bits which were also present on the 8096 are the Transmit Interrupt (TI) bit, the Receive Interrupt (RI) bit, and the Received Bit 8 (RB8) or Receive Parity Error (RPE) bit. SP_STAT is read-only in Window 0 and is shown in Figure 10-1.

The receiver on the 80C196KB checks for a valid stop bit. If a stop bit is not found within the appropriate time, the Framing Error (FE) bit is set. When the stop bit is detected, the data in the receive shift register is

SP_CON:	7	6	5	4	3	2	1	0	
	X	X	X	TB8	REN	PEN	M2	M1	11H

TB8 — Sets the ninth data bit for transmission. Cleared after each transmission. Not valid if parity is enabled.

REN — Enables the receiver

PEN — Enables the Parity function (even parity)

M2, M1 — Sets the mode. Mode0 = 00, Mode1 = 01, Mode2 = 10, Mode3 = 11

SP_STAT	7	6	5	4	3	2	1	0	
	RB8/ RPE	RI	TI	FE	TXE	OE	X	X	11H

RB8 — Set if the 9th data bit is high on reception (parity disabled)

RPE — Set if parity is enabled and a parity error occurred

RI — Set after the last data bit is sampled

TI — Set at the beginning of the STOP bit transmission

FE — Set if no STOP bit is found at the end of a reception

Figure 10-1. Serial Port Control and Status Registers

loaded into SBUF(RX) and the RI bit is set. If this happens before the previous byte in SBUF(RX) is read, the Overflow Error (OE) bit is set. The data in SBUF(RX) will always be the latest byte received; it will never be a combination of the two bytes. The RI, OE, and FE bits are reset when SP_STAT is read.

The Transmitter Empty (TXE) bit is set if the transmit buffer is empty and ready to take up to two characters. TXE gets cleared as soon as a byte is written to SBUF. Two bytes may be written consecutively to SBUF if TXE is set. One byte may be written if TI alone is set. By definition, if TXE has just been set, a transmission has completed and TI will be set. The TI bit is reset when the CPU reads the SP_STAT registers.

The TB8 bit is cleared after each transmission and both TI and RI are cleared when SP_STAT read. The RI and TI status bits can be set by writing to SP_STAT in window 15 but they will not cause an interrupt. Reading of SP_CON in Window 15 will read the last value written. Whenever the TXD pin is used for the serial port it must be enabled by setting IOC1.5 to a 1. I/O control register 1 can be read in window 15 to determine the setting.

STARTING TRANSMISSIONS AND RECEPTIONS

In Mode 0, if REN = 1, writing to SBUF (TX) will start a transmission. Causing a rising edge on REN, or clearing RI with REN = 1, will start a reception. Setting REN = 0 will stop a reception in progress and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before RI is cleared. This can be handled in an interrupt environment by using software flags or in straight-line code by using the Interrupt Pending register to signal the completion of a reception.

In the asynchronous modes, writing to SBUF (TX) starts a transmission. A falling edge on RXD will begin a reception if REN is set to 1. New data placed in SBUF (TX) is held and will not be transmitted until the end of the stop bit has been sent.

In all modes, the RI flag is set after the last data bit is sampled approximately in the middle of the bit time. Also for all modes, the TI flag is set after the last data bit (either 8th or 9th) is sent, also in the middle of the bit time. The flags clear when SP_STAT is read, but do not have to be clear for the port to receive or transmit. The serial port interrupt bit is set as a logical OR of the RI and TI bits. Note that changing modes will reset the Serial Port and abort any transmission or reception in progress on the channel.

DETERMINING BAUD RATES

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. Reading or writing to this register in window 15 is reserved by Intel for future use. This register must be loaded sequentially with 2 bytes (least significant byte first). The serial port will not function between the loading of the first and second bytes. The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum input frequency is 3 MHz on T2CLK. This provides the needed synchronization to the internal serial port clocks.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4 * (B + 1)}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64 * (B + 1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16 * B}; \quad B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12 MHz, are shown below.

Baud Rate	Baud Register Value	
	Mode 0	Others
9600	8137H	8013H
4800	8270H	8026H
2400	84E1H	804DH
1200	89C3H	809BH
300	A70FH	8270H

The maximum baud rates are 3.0 Mbaud synchronous and 750 Kbaud asynchronous with 12 MHz on XTAL1.

10.2 Serial Port Interrupts

The serial port generates one of three possible interrupts: Transmit Interrupt TI(2030H), Receive Interrupt RI(2032H) and SERIAL(200CH). When the RI bit gets set an interrupt is generated through either 200CH or 2032H depending on which interrupt is enabled. INT_MASK1.1 controls the serial port receive interrupt through location 2032H and INT_MASK.6 controls serial port interrupts through location 2032H. The 8096 shared the TI and RI interrupts on the SERIAL interrupt vector. On the 80C196KB, these interrupts share both the serial interrupt vector and have their own interrupt vectors.

When the TI bit is set it can cause an interrupt through the vectors at locations 200CH or 2032H. Interrupt through location 2032H is determined by INT_MASK1.0. Interrupts through the serial interrupt is controlled by the same bit as the RI interrupt(INT_MASK.6). The user should not mask off the serial port interrupt when using the double-buffered feature of the transmitter, as it could cause a missed count in the number of bytes being transmitted.

10.3 Serial Port Modes

MODE 0

Mode 0 is a synchronous mode which is commonly used for shift register based I/O expansion. In this mode the TXD pin outputs a set of 8 pulses while the RXD pin either transmits or receives data. Data is transferred 8 bits at a time with the LSB first. A diagram of the relative timing of these signals is shown in Figure 10-2. Note that this is the only mode which uses RXD as an output.

Mode 0 Timings

In Mode 0, the TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 10-2 shows the waveforms and timing. Note that the port starts functioning when a '1' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port expands the I/O capability of the 80C196KB by simply adding shift registers. A schematic of a typical circuit is shown in Figure 10-3. This circuit inverts the data coming in, so it must be reinverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

MODE 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode is shown in Figure 10-4. It consists of 10 bits; a start bit (0), 8 data bits (LSB first), and a stop bit (1). If parity is enabled by setting SPCON.2, an even parity bit is sent instead of the 8th data bit and parity is checked on reception.

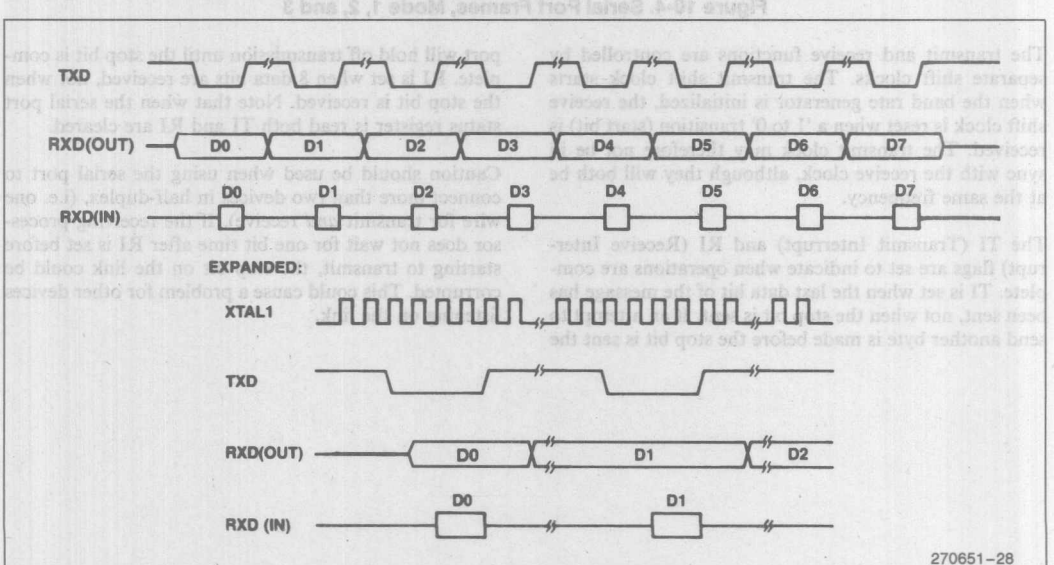


Figure 10-2. Mode 0 Timing

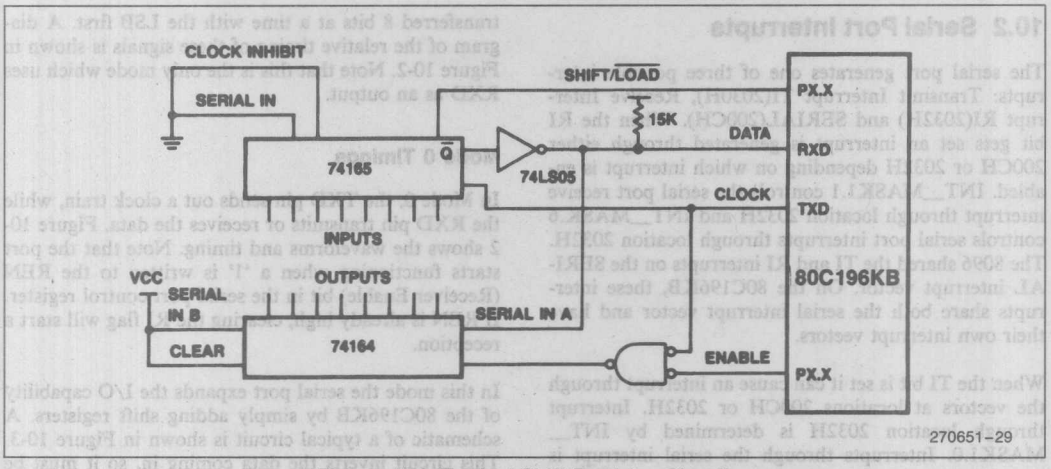


Figure 10-3. Typical Shift Register Circuit

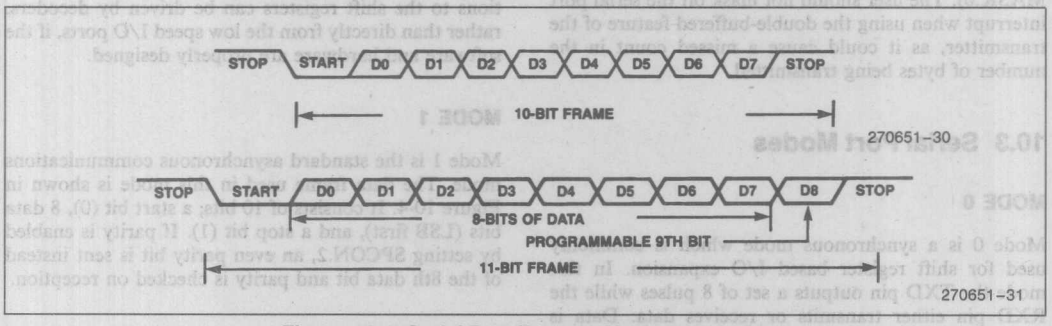


Figure 10-4. Serial Port Frames, Mode 1, 2, and 3

The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

Caution should be used when using the serial port to connect more than two devices in half-duplex, (i.e. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be corrupted. This could cause a problem for other devices listening on the link.

MODE 2

Mode 2 is the asynchronous 9th bit recognition mode. This mode is commonly used with Mode 3 for multiprocessor communications. Figure 10-4 shows the data frame used in this mode. It consists of a start bit (0), 9 data bits (LSB first), and a stop bit (1). When transmitting, the 9th bit can be set to a one by setting the TB8 bit in the control register before writing to SBUF (TX). The TB8 bit is cleared on every transmission, so it must be set prior to writing to SBUF (TX). During reception, the serial port interrupt and the Receive Interrupt (RI) bit will not be set unless the 9th bit being received is set. This provides an easy way to have selective reception on a data link. Parity cannot be enabled in this mode.

MODE 3

Mode 3 is the asynchronous 9th bit mode. The data frame for this mode is identical to that of Mode 2. The transmission differences between Mode 3 and Mode 2 are that parity can be enabled (PEN = 1) and cause the 9th data bit to take the even parity value. The TB8 bit can still be used if parity is not enabled (PEN = 0). When in Mode 3, a reception always causes an interrupt, regardless of the state of the 9th bit. The 9th bit is stored if PEN = 0 and can be read in bit RB8. If PEN = 1 then RB8 becomes the Receive Parity Error (RPE) flag.

Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of Mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit rather than the 8th. The 9th bit can be used for parity or multiple processor communications.

10.4 Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In Mode 2 if the received 9th data bit is zero, the serial port interrupt is not activated. In Mode 3, the serial port interrupt is activated regardless of the value in the 9th bit. The way to use this feature in multiprocessor systems is described below.

The master processor is set to Mode 3 so it always gets interrupts from serial receptions. The slaves are set in Mode 2 so they only have receive interrupts if the 9th bit is set. Two types of frames are used: address frames which have the 9th bit set and data frames which have the 9th bit cleared. When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. Slaves in Mode 2 will not be interrupted by a data frame, but an address frame will interrupt all slaves. Each slave can examine the received byte and see if it is being addressed. The addressed slave switches to Mode 3 to receive the coming data frames, while the slaves that were not addressed stay in Mode 2 continue executing.

11.0 A/D CONVERTER

Analog Inputs to the 80C196KB System are handled by the A/D converter System. As shown in Figure 11-4, the converter system has an 8 channel multiplexer, a sample-and-hold, and a 10 bit successive approximation A/D converter. Conversions can be performed on one of eight channels, the inputs of which share pins with port 0. A conversion can be done in as little as 91 state times.

Conversions are started by loading the AD_COMMAND register at location 02H with the channel number. The conversion can be started immediately by setting the GO bit to a one. If it is cleared the conversion will start when the HSO unit triggers it. The result of the conversion is read in the AD_RESULT(High) and AD_RESULT(Low) registers. The AD_RESULT(High) contains the most significant eight bits of the conversion. The AD_RESULT(Low) register contains the remaining two bits and the A/D channel number and A/D status. The format for the AD_COMMAND register is shown in Figure 11-1. In Window 15, reading the AD_COMMAND register will read the last command written. Writing to the AD_RESULT register will write a value into the result register.

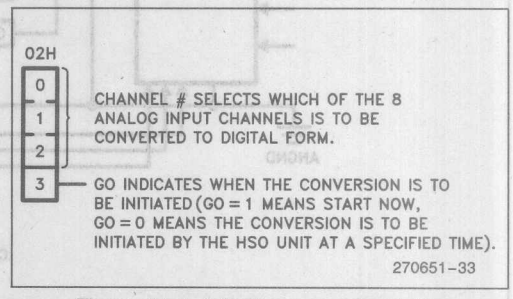


Figure 11-1. A/D Command Register

The A/D converter can cause an interrupt through the vector at location 2002H when it completes a conversion. It is also possible to use a polling method by checking the Status (S) bit in the lower byte of the AD_RESULT register, also at location 02H. The status bit will be a 1 while a conversion is in progress. It takes 8 state times to set this bit after a conversion is started. The upper byte of the result register contains the most significant 8 bits of the conversion. The lower byte format is shown in Figure 11-2.

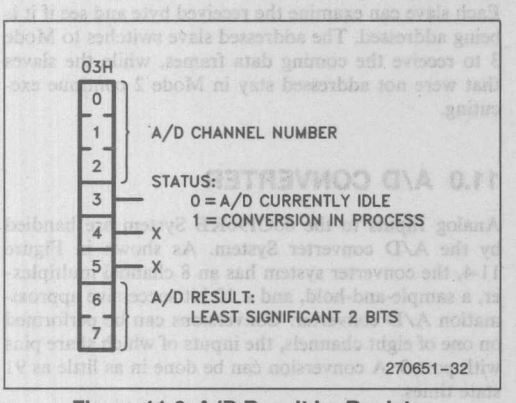


Figure 11-2. A/D Result Lo Register

At high crystal frequencies, more time is needed to allow the comparator to settle. For this reason IOC2.4 is provided to adjust the speed of the A/D conversion by disabling/enabling a clock prescaler.

A summary of the conversion time for the two options is shown below. The numbers represent the number of state times required for conversion, e.g., 91 states is 22.7 μs with an 8 MHz XTAL1 (providing a 250 ns state time.)

Clock Prescaler On IOC2.4 = 0	Clock Prescaler Off IOC2.4 = 1
158 States	91 States
26.33 μs @ 12 MHz	22.75 μs @ 8 MHz.
	18.2 μs @ 10 MHz

Figure 11-3. A/D Conversion Times

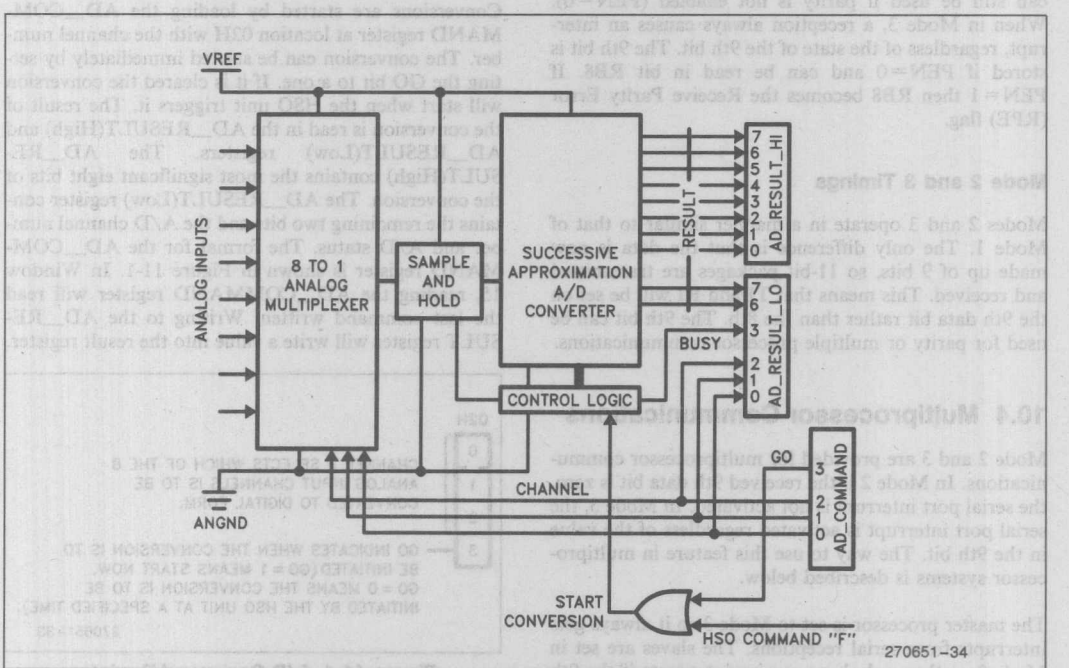


Figure 11-4. A/D Converter Block Diagram

11.1 A/D Conversion Process

The conversion process is initiated by the execution of HSO command 0FH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to the A/D converter control logic. If an HSO command was used, the conversion process will begin when Timer1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer1 delays will occur with a variance of about ±50 ns (assuming a stable clock on XTAL1). However, conversions initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution resulting in a variance of about 0.50 μs (XTAL1 = 12 MHz).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (Sample Delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for 8 state times in fast mode or 15 state times for slow mode (Sample Time). After this 8/15 state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while the conversion is in progress. The comparator is then autozeroed and the conversion begins. The sample delay and sample time uncertainties are each approximately ±50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the 80C196KB implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps (V_{REF} = 5.12V), while capacitive coupling creates 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result.

A successive approximation conversion is performed by comparing a sequence of reference voltages, to the analog input, in a binary search for the reference voltage that most closely matches the input. The ½ full scale reference voltage is the first tested. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111.1111.11b). If the analog input was less than the test voltage, bit 10 of the SAR is left a zero, and a new test voltage of ¼ full scale (0011.1111.11b) is tried. If this test voltage was lower than the analog input, bit 9 of the SAR is set and bit 8 is cleared for the next test (0101.1111.11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in the SAR where it can be read by software.

The total number of state times required for a conversion is determined by the setting of IOC2.4 clock prescaler bit. With the bit set the conversion time is 91 states for a 10-bit conversion and 158 states when the bit is cleared.

11.2 A/D Interface Suggestions

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's design, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

For the 80C196KB, these factors are idealized in Figure 11-5. The external input circuit must be able to charge a sample capacitor (C_S) through a series resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 80C196KB, C_S is around 2 pF, R_I is around 5 KΩ and I_L is specified as 3 μA maximum. In determining the necessary source impedance R_S, the value of V_{BIAS} is not important.

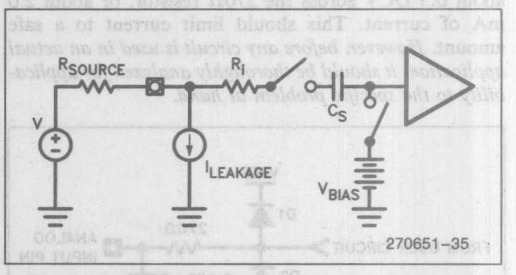


Figure 11-5. Idealized A/D Sampling Circuitry

External circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.61 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ can result in an external error of at least one LSB due to the voltage drop caused by the 1 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μs (12 MHz clock) sample window.

It is important to note that source impedance requirements relax if an external capacitor of sufficient size is attached directly to the analog input pin. Since the internal sample capacitor is around 2.0 pF, an external 0.005 μF capacitor (2048 × 2.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C. leakage caused by the

from a 0.005 μF capacitor in 1 μs. Therefore, the capacitor connected externally to the pin should be at least 0.005 μF if the source impedance is too large to provide the needed accuracy on its own.

Placing an external capacitor on each analog input will also reduce the sensitivity to noise, as the capacitor combines with series resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to the external capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 11-6 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input. Should the input voltage inappropriately drop significantly below ground, diode D2 will forward bias at about 0.8 DCV. Since the specification of the pin has an absolute maximum low voltage rating of −0.3 DCV, this will leave about 0.5 DCV across the 270Ω resistor, or about 2.0 mA of current. This should limit current to a safe amount. *However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.*

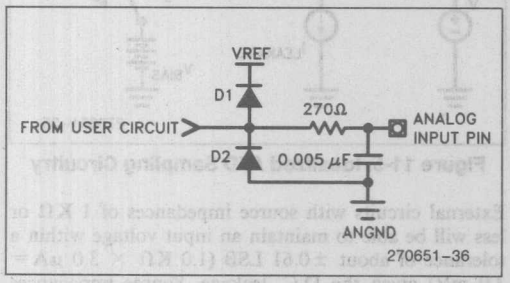


Figure 11-6. Suggested A/D Input Circuit

ANALOG REFERENCES

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to the two V_{SS} pins at the power supply. Bypass capacitors should also be used between V_{REF} and ANGND. ANGND should be within about a tenth of a volt of V_{SS}. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. See Section 13 for the minimum hardware connections.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC}. In addition, V_{REF} and is not being used. Remember that Port 0 receives its power from the V_{REF} and ANGND pins even when it is used as digital I/O.

11.3 The A/D Transfer Function

The conversion result is a 10-bit ratiometric representation of the input voltage, so the numerical value obtained from the conversion will be:

$$\text{INT}\,[1023 \times (V_{IN} - \text{ANGND})/(V_{REF} - \text{ANGND})].$$

This produces a stair-stepped transfer function when the output code is plotted versus input voltage (see Figure 11-7). The resulting digital codes can be taken as simple ratiometric information, or they provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, closing a servo-loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error, zero offset, full-scale error, differential non-linearity, and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, converter temperature drift, V_{CC} rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching and random noise should be considered. Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. However, the various sub-components of error are important in many applications. These error components are described in Section 11.5 and in the text below where ideal and actual converters are compared.

An unavoidable error simply results from the conversion of a continuous voltage to an integer digital representation. This error is called quantizing error, and is always ±0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 11-7 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 11-7 the Ideal Characteristic possesses unique qualities: it's first code transition occurs when the input voltage is 0.5 LSB; it's full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and it's code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

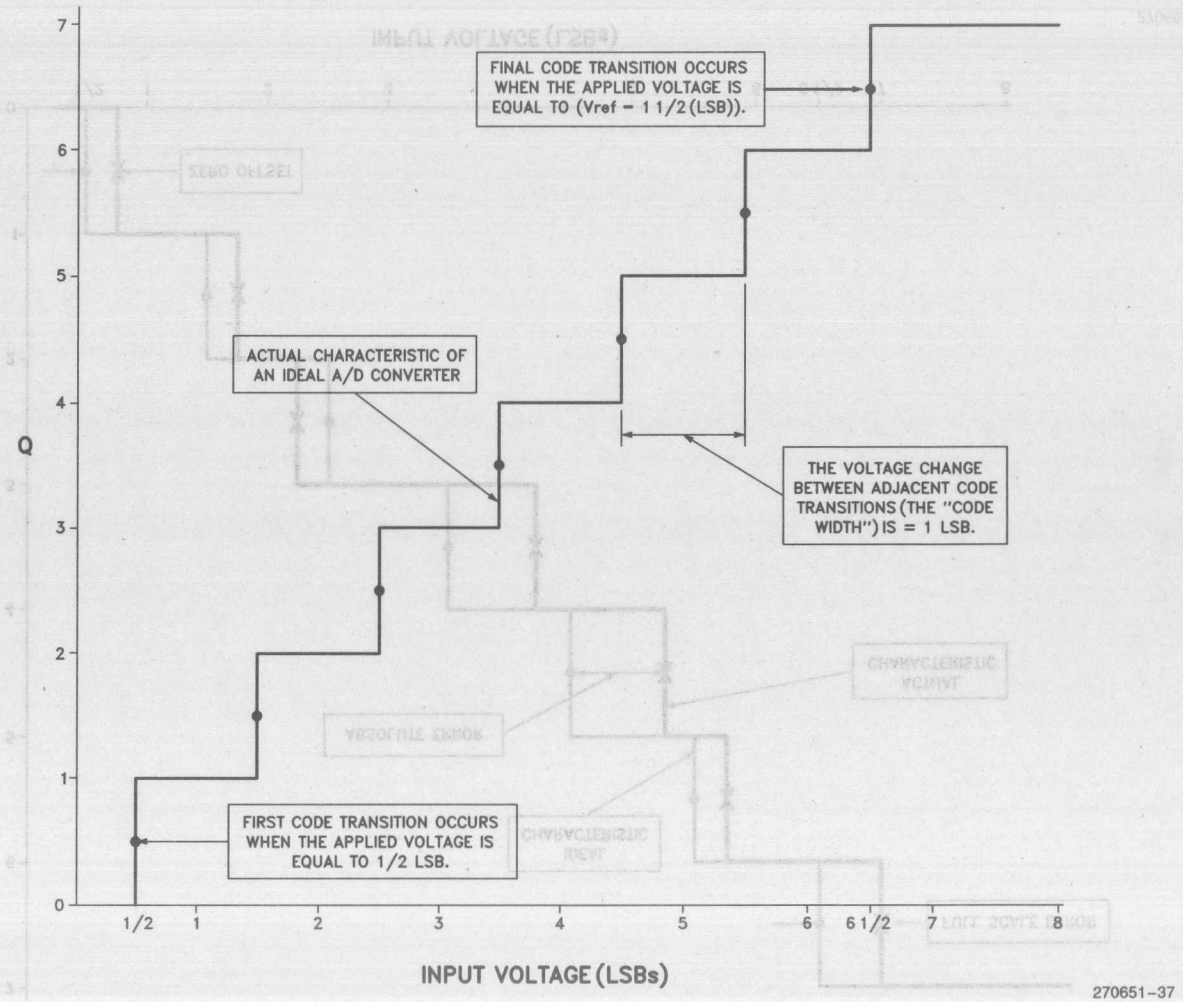


Figure 11-7. Ideal A/D Characteristic

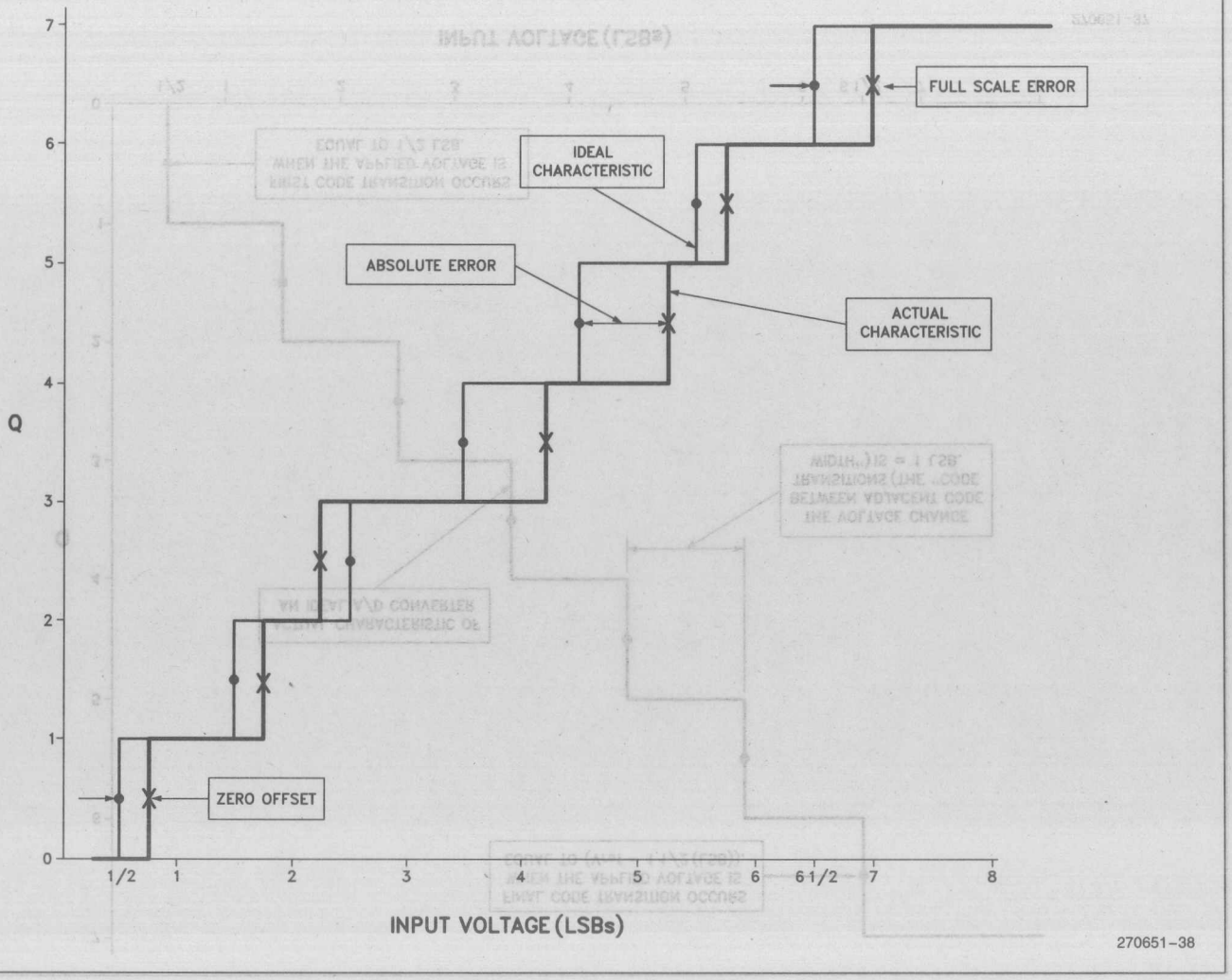


Figure 11-8. Actual and Ideal Characteristics

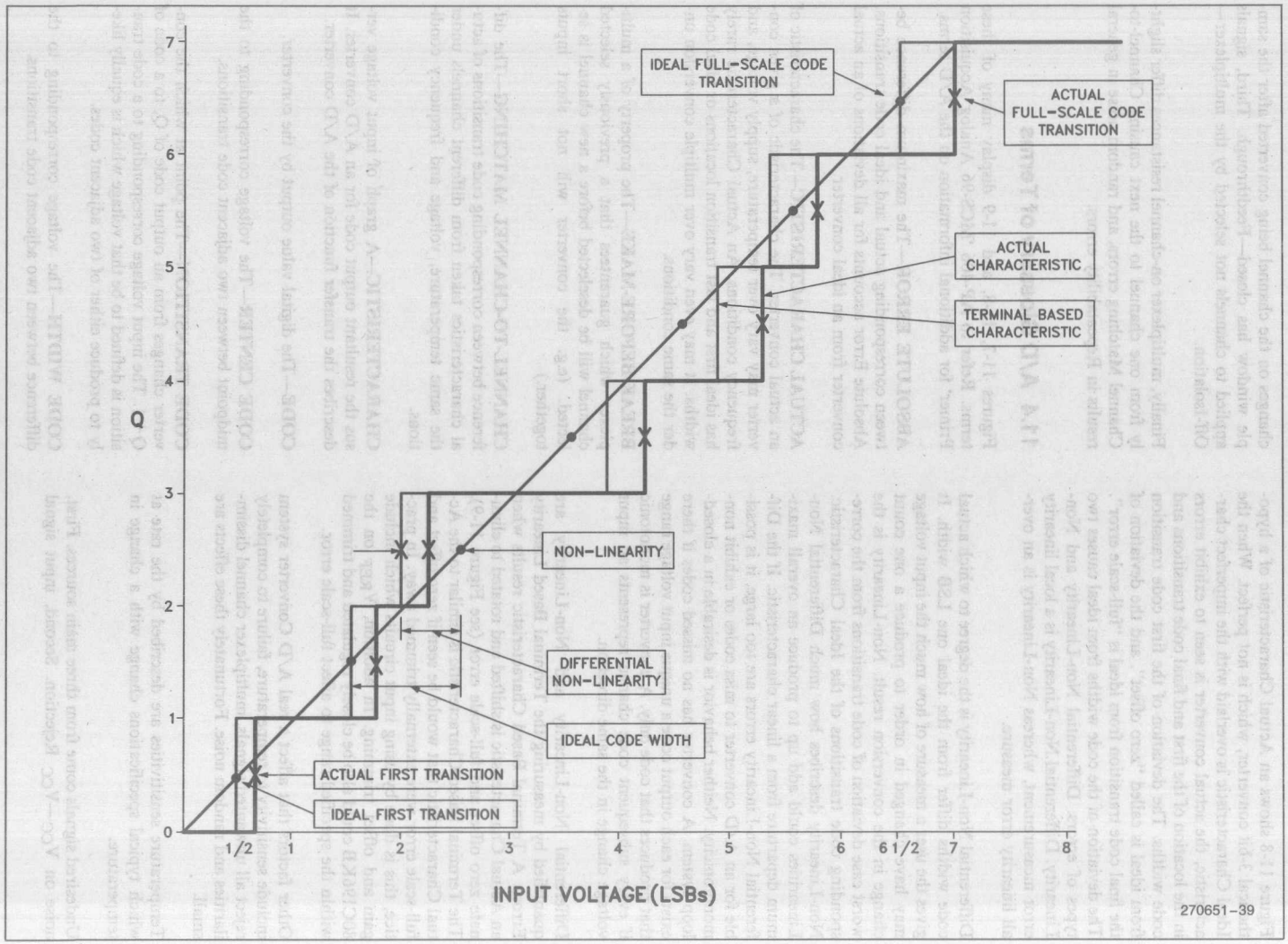


Figure 11-9. Terminal Based Characteristic

Figure 11-8 shows an Actual Characteristic of a hypothetical 3-bit converter, which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset", and the deviation of the final code transition from ideal is "full-scale error". The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measurement, whereas Non-Linearity is an overall linearity error measure.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal one LSB width. It gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result. Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add up to produce an overall maximum departure from a linear characteristic. If the Differential Non-Linearity errors are too large, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desirable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and rotated to eliminate zero offset and full-scale error (see Figure 11-9). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. In addition, V_{REF} on the 80C196KB could also be closely regulated and trimmed within the specified range to affect full-scale error.

Other factors that affect a real A/D Converter system include sensitivity to temperature, failure to completely reject all unwanted signals, multiplexer channel dissimilarities and random noise. Fortunately these effects are small.

Temperature sensitivities are described by the rate at which typical specifications change with a change in temperature.

Undesired signals come from three main sources. First, noise on V_{CC}—V_{CC} Rejection. Second, input signal changes on the channel being converted after the sample window has closed—Feedthrough. Third, signals applied to channels not selected by the multiplexer—Off-Isolation.

Finally, multiplexer on-channel resistances differ slightly from one channel to the next causing Channel-to-Channel Matching errors, and random noise in general results in Repeatability errors.

11.4 A/D Glossary of Terms

Figures 11-7, 11-8, and 11-9 display many of these terms. Refer to AP-406 'MCS-96 Analog Acquisition Primer' for additional information on the A/D terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversion under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—LEAST SIGNIFICANT BIT: The voltage value corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSBs, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristics.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which as been rotated and translated to remove zero offset and full-scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

12.0 I/O PORTS

There are five 8-bit I/O ports on the 80C196KB. Some of these ports are input only, some are output only, some are bidirectional and some have alternate functions. In addition to these ports, the HSI/O unit provides extra I/O lines if the timer related features of these lines are not needed.

Port 0 is an input port which is also used as the analog input for the A/D converter. Port 0 is read at location 0EH. Port 1 is a quasi-bidirectional port and is read or written to through location 0FH. Port 2 contains three types of port lines: quasi-bidirectional, input and output. Port2 is read or written from location 10H. The ports cannot be read or written in Window 15. The

input and output lines are shared with other functions in the 80C196KB as shown in Figure 12-1. Ports 3 and 4 are open-drain bidirectional ports which share their pins with the address/data bus. On EPROM and ROM parts, Port 3 and 4 are read and written through location 1FFEH.

PIN	FUNC.	ALTERNATE FUNCTION	CONTROL REG.
2.0	Output	TXD (Serial Port Transmit)	IOC1.5
2.1	Input	RXD (Serial Port Receive)	SPCON.3
2.3	Input	T2CLK (Timer2 Clock & Baud)	IOC0.7
2.4	Input	T2RST (Timer2 Reset)	IOC0.5
2.5	Output	PWM Output	IOC1.0
2.6	QBD*	Timer2 up/down select	IOC2.1
2.7	QBD*	Timer2 Capture	N/A

*QBD = Quasi-bidirectional

Figure 12-1. Port 2 Multiple Functions

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the data sheet that corresponds to the part being used.

12.1 Input Ports

Input ports and pins can only be read. There are no output drivers on these pins. The input leakage of these pins is in the microamp range. The specific values can be found in the data sheet for the device being considered.

The high impedance input pins on the 80C196KB have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pF.

In addition to acting as a digital input, each line of Port 0 can be selected to be the input of the A/D converter as discussed in Section 11. The pins on Port 0 are tested to have D.C. leakage of 3 microamps or less, as specified in the data sheet for the device being considered. The capacitance on these pins is approximately 1 pF and will instantaneously increase by around 2 pF when the pin is being sampled by the A/D converter.

The 80C196KB samples the input to the A/D for 8 state times in with the A/D clock prescaler off and 15 states with the A/D clock prescaler on. Details on the A/D converter can be found in Section 11.

Port 0 pins are special in that they may individually be used as digital inputs and analog inputs at the same time. A Port 0 pin being used as a digital input acts as the high impedance input ports just described. However, Port 0 pins being used as analog inputs are required to provide current to the internal sample capacitor when a conversion begins. This means that the input characteristics of a pin will change if a conversion is being done on that pin. In either case, if Port 0 is to be used as analog or digital I/O, it will be necessary to provide power to this port through the V_{REF} pin and ANGND pins.

Port 0 is only sampled when the SFR is read to reduce the noise in the A/D converter. The data must be stable one state time before the SFR is read.

12.2 Quasi-Bidirectional Ports

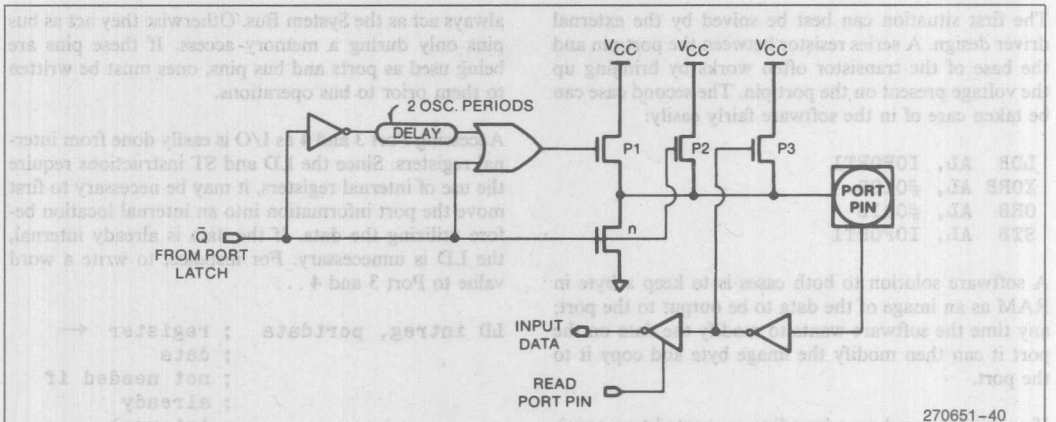
Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2 and is similar to the HSI in that the sample occurs during PH1 or during CLKOUT low. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pullup will remain on for one state time after the change.

Port 1, Port 2.6 and Port 2.7 are quasi-bidirectional ports. When the processor writes to the pins of a quasi-bidirectional port it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to its associated SFR bit, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pullup device which can be easily driven down by the device driving the input.

If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port.

Particular care should be exercised when using XOR opcodes or any opcode which is a read-modify-write instruction. It is possible for a Quasi-Bidirectional Pin to be written as a one, but read back as a zero if an external device (i.e., a transistor base) is pulling the pin below V_{IH}.

Quasi-bidirectional pins can be used as input and output pins without the need for a data direction register. They output a strong low value and a weak high value. The weak high value can be externally pulled low providing an input function. Figure 12-2 shows the configuration of a CHMOS quasi-bidirectional port.



CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after $\overline{Q}$ makes a 0-to-1 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 12-2. CHMOS Quasi-Bidirectional Port Circuit

Outputting a 0 on a quasi-bidirectional pin turns on the strong pull-down and turns off all of the pull-ups. When a 1 is output the pull-down is turned off and 3 pull-ups (strong-P1, weak-P3, very weak-P2) are turned on. Each time a pin switches from 0 to 1 transistor P1 turns on for two oscillator periods. P2 remains on until a zero is written to the pin. P3 is used as a latch, so it is turned on whenever the pin is above the threshold value (around 2 volts).

To reduce the amount of current which flows when the pin is externally pulled low, P3 is turned off when the pin voltage drops below the threshold. The current required to pull the pin from a high to a low is at its maximum just prior to the pull-up turning off. An external driver can switch these pins easily. The maximum current required occurs at the threshold voltage and is approximately 700 microamps.

HARDWARE CONNECTION HINTS

When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground from each pin is typically 7 mA or more. Therefore, if all 8 pins are tied to ground, 56 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

This potential problem can be solved in hardware or software. In software, never write a zero to a pin being used as an input.

In hardware, a 1K resistor in series with each pin will limit current to a reasonable value without impeding the ability to override the high impedance pullup. If all 8 pins are tied together a 120Ω resistor would be reasonable. The problem is not quite as severe when the inputs are tied to electronic devices instead of switches, as most external pulldowns will not hold 20 mA to 0.0 volts.

Writing to a Quasi-Bidirectional Port with electronic devices attached to the pins requires special attention. Consider using P1.0 as an input and trying to toggle P1.1 as an output:

```
ORB  IOPORT1, #00000001B ; Set P1.0
                         ; for input
XORB IOPORT1, #00000010B ; Complement
                         ; P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 80C196KB it is possible that it is being held low externally. This typically happens when the port pin drives the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7V. The 80C196KB will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin's SFR and the pin will not toggle.

The second problem, which is related to the first, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction, then the XORB will write a zero to P1.0 and it will no longer be useable as an input.

driver design. A series resistor between the port pin and the base of the transistor often works by bringing up the voltage present on the port pin. The second case can be taken care of in the software fairly easily:

```
LDB  AL, IOPORT1
XORB AL, #010B
ORB  AL, #001B
STB  AL, IOPORT1
```

A software solution to both cases is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and copy it to the port.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pullup resistor is recommended to reduce the possibility of noise glitches and to decrease the rise time of the line. On extremely long lines that are handling slow signals, a capacitor may be helpful in addition to the resistor to reduce noise.

12.3 Output Ports

Output pins include the bus control lines, the HSO lines, and some of Port 2. These pins can only be used as outputs as there are no input buffers connected to them. The output pins are output before the rising edge of PH1 and is valid some time during PH1. Externally, PH1 corresponds to CLKOUT low. It is not possible to use immediate logical instructions such as XOR PORT2, #00111B to toggle these pins.

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: $\overline{\text{TXD}}$, $\overline{\text{RXD}}$ (in Mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to $V_{CC}-0.3$ volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 3.2 mA at 0.45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 12-3 shows the internal configuration of a bus pin.

12.4 Ports 3 and 4/AD0–15

These pins have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. If the $\overline{\text{EA}}$ line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Accessing Port 3 and 4 as I/O is easily done from internal registers. Since the LD and ST instructions require the use of internal registers, it may be necessary to first move the port information into an internal location before utilizing the data. If the data is already internal, the LD is unnecessary. For instance, to write a word value to Port 3 and 4 . . .

```
LD intreg, portdata   ; register ←
                      ; data
                      ; not needed if
                      ; already
                      ; internal

ST intreg, 1FFEH      ; register →
                      ; Port 3 and 4
```

To read Port 3 and 4 requires that "ones" be written to the port registers to first setup the input port configuration circuit. Note that the ports are reset to this input condition, but if zeroes have been written to the port, then ones must be re-written to any pins which are to be used as inputs. Reading Port 3 and 4 from a previously written zero condition is as follows . . .

```
LD intregA, #0FFFFH   ; setup port
                      ; change mode
                      ; pattern
ST intregA, 1FFEH     ; register →
                      ; Port 3 and 4
                      ; LD & ST not
                      ; needed if
                      ; previously
                      ; written as ones

LD intregB, 1FFEH     ; register ←
                      ; Port 3 and 4
```

Note that while the format of the LD and ST instructions are similar, the source and destination directions change.

When acting as the system bus the pins have strong drivers to both V_{CC} and V_{SS}. These drivers are used whenever data is being output on the system bus and are not used when data is being output by Ports 3 and 4. The pins, external input buffers and pulldowns are shared between the bus and the ports. The ports use different output buffers which are configured as open-drain, and require external pullup resistors. (open-drain is the MOS version of open-collector.) The port pins and their system bus functions are shown in Figure 12-3.

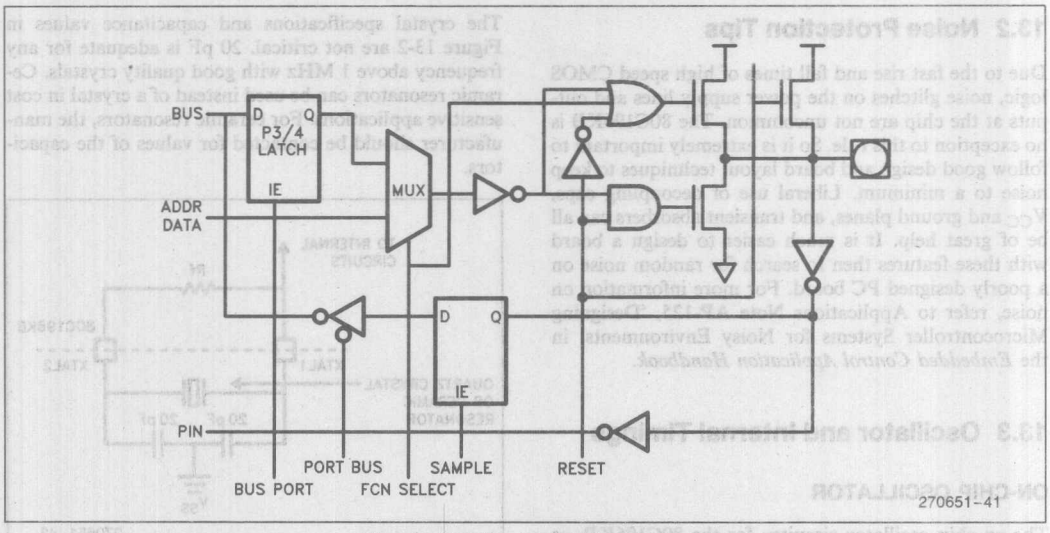


Figure 12-3. Port 3,4 AD0-15 Pins

Ports 3 and 4 on the 80C196KB are open drain ports. There is no pullup when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to Ports 3 and 4 and the bus pins is shown in Figure 12-3.

When Ports 3 and 4 are to be used as inputs, or as bus pins, they must first be written with a '1'. This will put the ports in a high impedance mode. When they are used as outputs, a pullup resistor must be used externally. The sink capability of these pins is on the order of 3.2 milliamps so the total pullup current to the pin must be less than this. A 15K pullup resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drains, their structure is different when they are used as part of the bus.

Port 3 and 4 can be reconstructed as I/O ports from the Address/Data bus. Refer to Section 16.7 for details.

13.0 MINIMUM HARDWARE CONSIDERATIONS

The 80C196KB requires several external connections to operate correctly. Power and ground must be connected, a clock source must be generated, and a reset circuit must be present. We will look at each of these areas in detail.

13.1 Power supply

Power to the 80C196KB flows through 5 pins. V_{CC} supplies the positive voltage to the digital portion of the chip while V_{REF} supplies the A/D converter and Port0 with a positive voltage. These two pins need to be connected to a 5 volt power supply. When using the A/D converter, it is desirable to connect V_{REF} to a separate power supply, or at least a separate trace to minimize the noise in the A/D converter.

The three common return pins, $V_{SS}1$, $V_{SS}2$, and Angd, must all be nominally at 0 volts. $V_{SS}1$ and $V_{SS}2$ should be connected with a short as lead as possible to minimize the voltage difference between them. Digital and Analog ground should be connected together at the power supply. Even if the A/D converter is not being used, V_{REF} and Angd must still be connected for Port0 to function. The maximum current drain of the 80C196KB is about 55 mA at 12 Mhz.

13.2 Noise Protection Tips

Due to the fast rise and fall times of high speed CMOS logic, noise glitches on the power supply lines and outputs at the chip are not uncommon. The 80C196KB is no exception to this rule. So it is extremely important to follow good design and board layout techniques to keep noise to a minimum. Liberal use of decoupling caps, V_{CC} and ground planes, and transient absorbers can all be of great help. It is much easier to design a board with these features then to search for random noise on a poorly designed PC board. For more information on noise, refer to Applications Note AP-125, 'Designing Microcontroller Systems for Noisy Environments' in the *Embedded Control Application Handbook.*

13.3 Oscillator and Internal Timings

ON-CHIP OSCILLATOR

The on-chip oscillator circuitry for the 80C196KB, as shown in Figure 13.1, consists of a crystal-controlled, positive reactance oscillator. In this application, the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

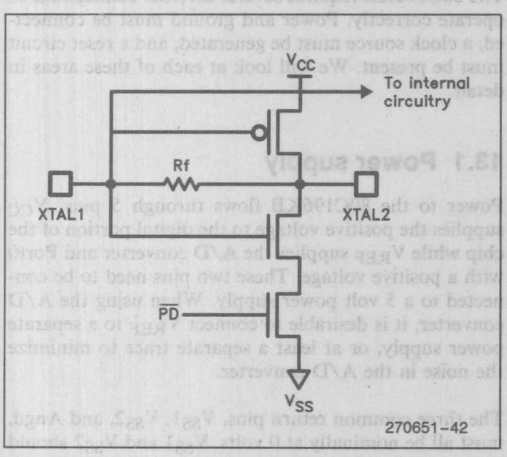


Figure 13-1. On-chip Oscillator Circuitry

The feedback resistor, Rf, consists of paralleled n-channel and p-channel FETs controlled by the PD (powerdown) bit. Rf acts as an open when in Powerdown Mode. Both XTAL1 and XTAL2 also have ESD protection on the pins which is not shown in the figure.

The crystal specifications and capacitance values in Figure 13-2 are not critical. 20 pF is adequate for any frequency above 1 MHz with good quality crystals. Ceramic resonators can be used instead of a crystal in cost sensitive applications. For ceramic resonators, the manufacturer should be contacted for values of the capacitors.

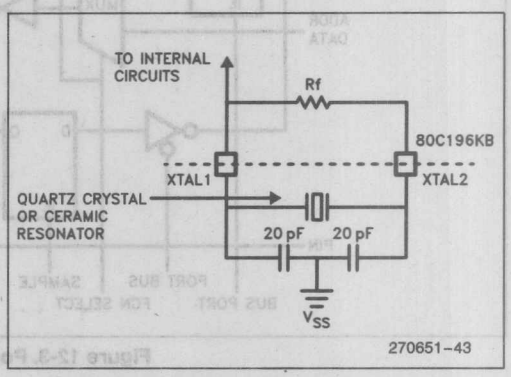


Figure 13-2. External Crystal Connections

INTERNAL TIMINGS

Internal operation of the chip is based on the oscillator frequency divided by two, giving the basic time unit, known as a 'state time'. With a 12 Mhz crystal, a state time is 167 nS. Since the 80C196KB can operate at many frequencies, the times given throughout this overview will be in state times.

Two non-overlapping internal phases are created by the clock generator: phase 1 and phase 2 as shown in Figure 13-3. CLKOUT is generated by the rising edge of phase 1 and phase 2. This is not the same as the 8096BH, which uses a three phase clock. Changing from a three phase clock to a two phase one speeds up operation for a set oscillator frequency. Consult the latest data sheet for AC timing specifications.

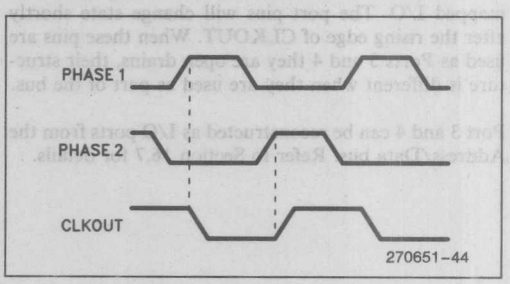


Figure 13-3. Internal Clock Phases

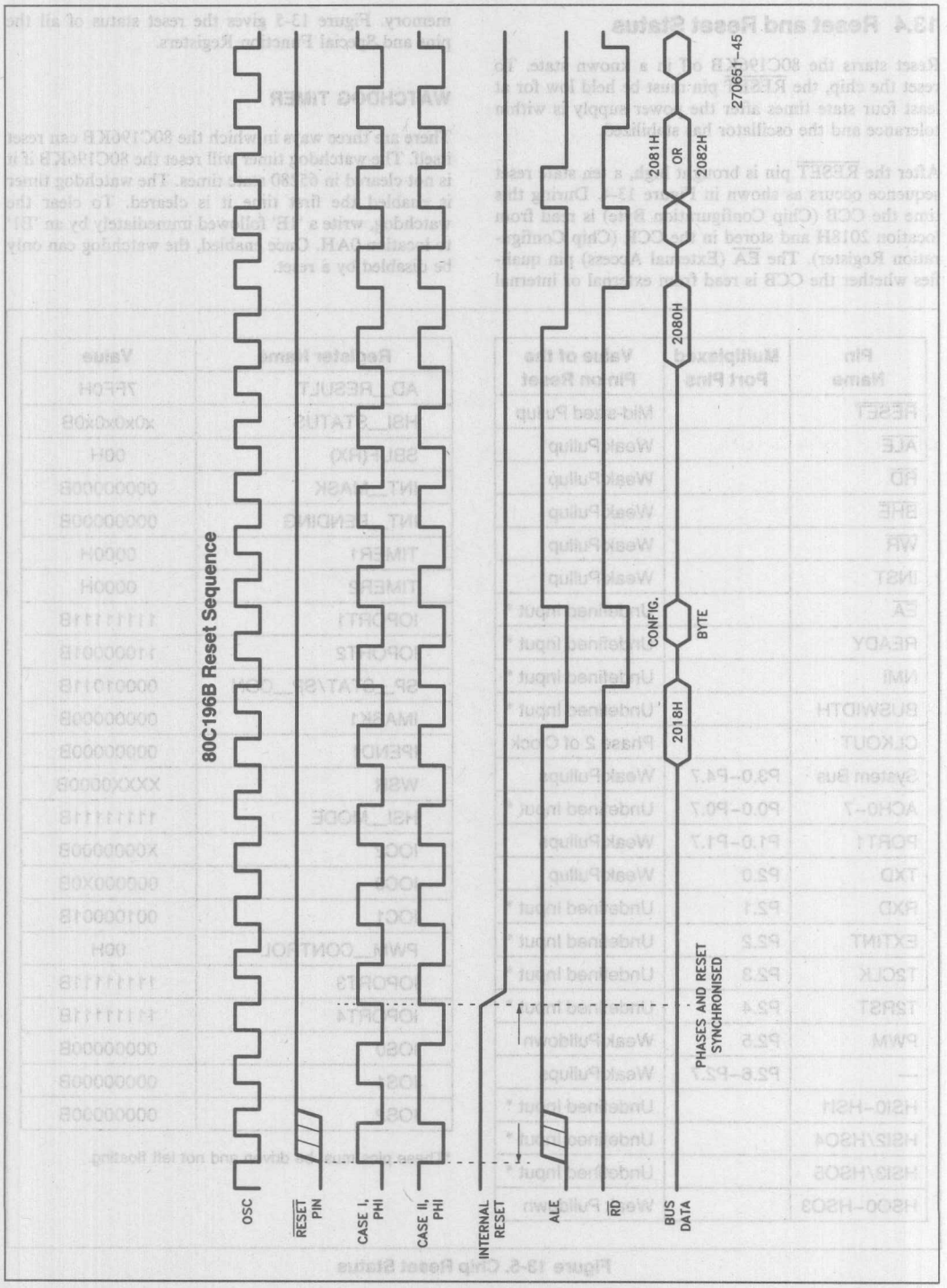


Figure 13-4. Reset Sequence

13.4 Reset and Reset Status

Reset starts the 80C196KB off in a known state. To reset the chip, the $\overline{\text{RESET}}$ pin must be held low for at least four state times after the power supply is within tolerance and the oscillator has stabilized.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence occurs as shown in Figure 13-4. During this time the CCB (Chip Configuration Byte) is read from location 2018H and stored in the CCR (Chip Configuration Register). The $\overline{\text{EA}}$ (External Access) pin qualifies whether the CCB is read from external or internal memory. Figure 13-5 gives the reset status of all the pins and Special Function Registers.

WATCHDOG TIMER

There are three ways in which the 80C196KB can reset itself. The watchdog timer will reset the 80C196KB if it is not cleared in 65280 state times. The watchdog timer is enabled the first time it is cleared. To clear the watchdog, write a '1E' followed immediately by an 'E1' to location 0AH. Once enabled, the watchdog can only be disabled by a reset.

Pin Name	Multiplexed Port Pins	Value of the Pin on Reset
$\overline{\text{RESET}}$		Mid-sized Pullup
$\overline{\text{ALE}}$		Weak Pullup
$\overline{\text{RD}}$		Weak Pullup
$\overline{\text{BHE}}$		Weak Pullup
$\overline{\text{WR}}$		Weak Pullup
INST		Weak Pullup
$\overline{\text{EA}}$		Undefined Input *
READY		Undefined Input *
NMI		Undefined Input *
BUSWIDTH		Undefined Input *
CLKOUT		Phase 2 of Clock
System Bus	P3.0–P4.7	Weak Pullups
ACH0–7	P0.0–P0.7	Undefined Input *
PORT1	P1.0–P1.7	Weak Pullups
TXD	P2.0	Weak Pullup
RXD	P2.1	Undefined Input *
EXTINT	P2.2	Undefined Input *
T2CLK	P2.3	Undefined Input *
T2RST	P2.4	Undefined Input *
PWM	P2.5	Weak Pulldown
—	P2.6–P2.7	Weak Pullups
HSI0–HSI1		Undefined Input *
HSI2/HSO4		Undefined Input *
HSI3/HSO5		Undefined Input *
HSO0–HSO3		Weak Pulldown

Register Name	Value
AD_RESULT	7FF0H
HSI_STATUS	x0x0x0x0B
SBUF(RX)	00H
INT_MASK	00000000B
INT_PENDING	00000000B
TIMER1	0000H
TIMER2	0000H
IOPORT1	11111111B
IOPORT2	11000001B
SP_STAT/SP_CON	00001011B
IMASK1	00000000B
IPEND1	00000000B
WSR	XXXX0000B
HSI_MODE	11111111B
IOC2	X0000000B
IOC0	000000X0B
IOC1	00100001B
PWM_CONTROL	00H
IOPORT3	11111111B
IOPORT4	11111111B
IOS0	00000000B
IOS1	00000000B
IOS2	00000000B

*These pins must be driven and not left floating.

Figure 13-5. Chip Reset Status

CLOCK DETECT ENABLE

The Clock Detect Enable circuit is activated by strapping the CDE pin to V_{CC} on the 80C196KB. The CDE pin will become an extra V_{SS} pin on future proliferations. To make an 80C196KB compatible with future parts, the CDE pin should be tied to V_{SS} or jumpered to V_{CC} and V_{SS}. When activated, the Clock Detect Enable circuit will reset the chip if the clock input falls below a specified frequency. At 6 volts V_{CC}, the frequency is about 250 Khz and at 4 volts is around 28 Khz.

RST INSTRUCTION

Executing a RST instruction will also reset the 80C196KB. The opcode for the RST instruction is 0FFH. By putting pullups on the Addr/data bus, unimplemented areas of memory will read 0FFH and cause the 80C196KB to be reset.

RESET CIRCUITS

The simplest way to reset an 80C196KB is to insert a capacitor between the $\overline{\text{RESET}}$ pin and V_{SS}. The 80C196KB has an internal pullup which has a value between 6K and 50K ohms. A 5 uF or greater capacitor should provide sufficient reset time as long as Vcc rises quickly.

Figure 13-6 shows what the $\overline{\text{RESET}}$ pin looks like internally. The $\overline{\text{RESET}}$ pin functions as an input and as an output to reset an entire system with a watchdog timer overflow, clock detect failure, or by executing a RST instruction. For a system reset application, the reset circuit should be a one-shot with an open collector output. The reset pulse may have to be lengthened and buffered since $\overline{\text{RESET}}$ is only asserted for four state times. If this is done, it is possible for the 80C196KB to start running before other chips in the system are out of reset. Software must take this condition into account. A capacitor cannot be connected directly to $\overline{\text{RESET}}$ if it is to drive the reset pins of other chips in the circuit. The capacitor may keep the voltage on the pin from going below guaranteed V_{IL} for circuits connected to the $\overline{\text{RESET}}$ pin. Figure 13-7 shows an example of a system reset circuit.

13.5 Minimum Hardware Connections

Figure 13-8 shows the minimum connections needed to get the 80C196KB up and running. It is important to tie all unused inputs to V_{CC} or V_{SS}. If these pins are left floating, they can float to a mid voltage level and draw excessive current. Some pins such as NMI or EXTINT may generate spurious interrupts if left unconnected.

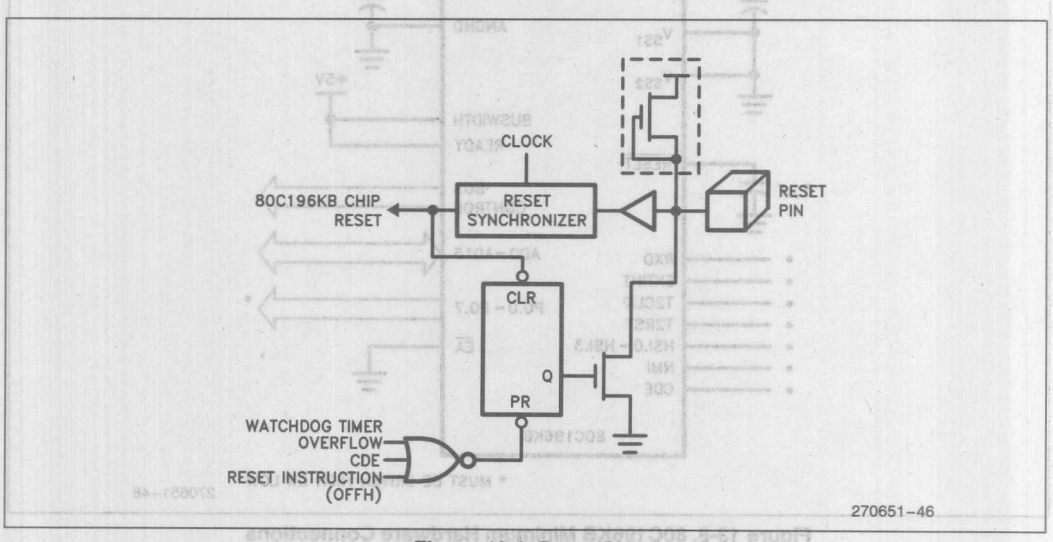
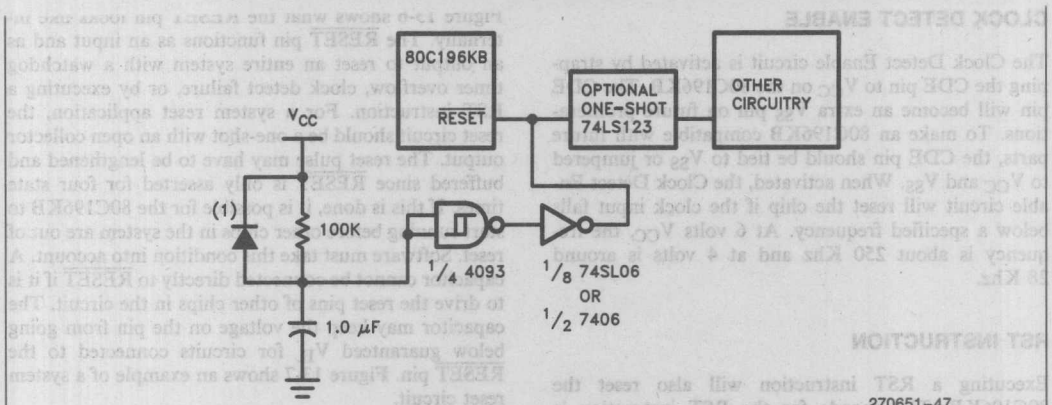


Figure 13-6. Reset Pin



NOTE:
1. The diode will provide a faster cycle time repetitive power-on-resets.

Figure 13-7. System Reset Circuit

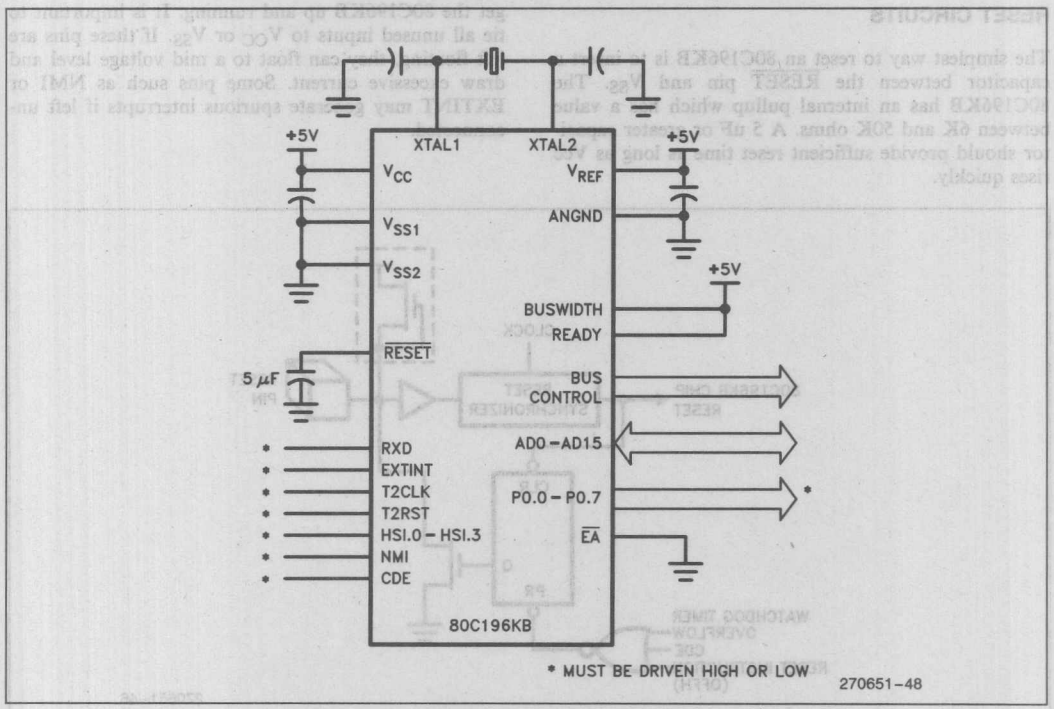


Figure 13-8. 80C196KB Minimum Hardware Connections

14.0 SPECIAL MODES OF OPERATION

The 80C196KB has Idle and Powerdown Modes to reduce the amount of current consumed by the chip. The 80C196KB also has an ONCE (ON-Circuit-Emulation) Mode to isolate itself from the rest of the components in the system.

14.1 Idle Mode

The Idle Mode is entered by executing the instruction 'IDLPD #1'. In the Idle Mode, the CPU stop executing. The CPU clocks are frozen at logic state zero, but the peripheral clocks continue to be active. CLKOUT continues to be active. Power consumption in the Idle Mode is reduced to about 40% of the active Mode.

The CPU exits the Idle Mode by any enabled interrupt source or a hardware reset. Since all of the peripherals are running, the interrupt can be generated by the HSI, HSO, A/D, serial port, etc. When an interrupt brings the CPU out of the Idle Mode, the CPU vectors to the corresponding interrupt service routine and begins executing. The CPU returns from the interrupt service routine to the next instruction following the 'IDLPD #1' instruction that put the CPU in the Idle Mode.

In the Idle Mode, the system bus control pins (ALE, $\overline{RD}$, $\overline{WR}$, INST, and $\overline{BHE}$), go to their inactive states. Ports 3 and 4 will retain the value present in their data latches if being used as I/O ports. If these ports are the ADDR/DATA bus, the pins will float.

It is important to note the Watchdog Timer continues to run in the Idle Mode if it is enabled. So the chip must be awakened every 64K state times to clear the Watchdog or the chip will reset.

14.2 Powerdown Mode

The Powerdown Mode is entered by executing the instruction, 'IDLPD #2'. In the Powerdown Mode, all internal clocks are frozen at logic state zero and the oscillator is shut off. All 232 bytes of registers and most peripherals hold their values if V_{CC} is maintained. Power is reduced to the device leakage and is in the uA range. The 87C196KB (EPROM part) will consume more power if the EPROM window is not covered.

In Powerdown, the bus control pins go to their inactive states. All of the output pins will assume the value in their data latches. Ports 3 and 4 will continue to act as ports in the single chip mode or will float if acting as the ADDR/DATA bus.

To prevent accidental entry into the Powerdown Mode, this feature may be disabled at reset by clearing bit 0 of the CCR (Chip Configuration Register). Since the default value of the CCR bit 0 is 1, the Powerdown Mode is normally enabled.

The Powerdown Mode can be exited by a chip reset or a transition on the external interrupt pin. If the $\overline{RESET}$ pin is used, it must be asserted long enough for the oscillator to stabilize.

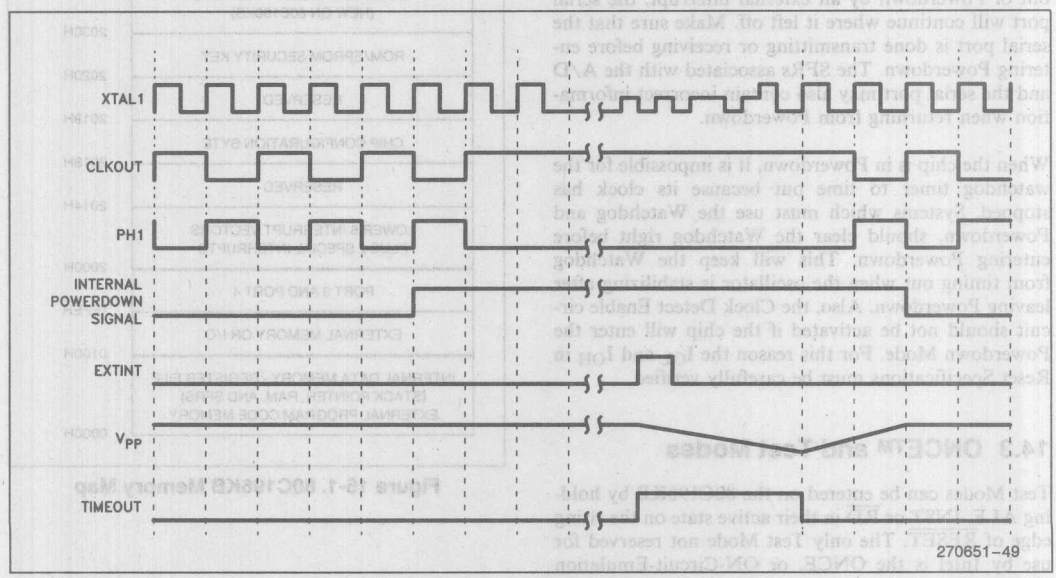


Figure 14-1. Power Up and Power Down Sequence

When exiting Powerdown with an external interrupt, a positive level on the pin mapped to INT7 (either EXTINT or port0.7) will bring the chip out of Powerdown Mode. The interrupt does not have to be unmasked to exit Powerdown. An internal timing circuit ensures that the oscillator has time to stabilize before turning on the internal clocks. Figure 14-1 shows the power down and power up sequence using an external interrupt.

During normal operation, before entering Powerdown Mode, the V_{PP} pin will rise to V_{CC} through an internal pullup. The user must connect a capacitor between V_{PP} and V_{SS}. A positive level on the external interrupt pin starts to discharge this capacitor. The internal current source that discharges the capacitor can sink approximately 100 uA. When the voltage goes below about 1 volt on the V_{PP} pin, the chip begins executing code. A 1uF capacitor would take about 4 ms to discharge to 1 volt.

If the external interrupt brings the chip out of Powerdown, the corresponding bit will be set in the interrupt pending register. If the interrupt is unmasked, the part will immediately execute the interrupt service routine, and return to the instruction following the IDLPD instruction that put the chip into Powerdown. If the interrupt is masked, the chip will start at the instruction following the IDLPD instruction. The bit in the pending register will remain set, however.

All peripherals should be in an inactive state before entering Powerdown. If the A/D converter is in the middle of a conversion, it is aborted. If the chip comes out of Powerdown by an external interrupt, the serial port will continue where it left off. Make sure that the serial port is done transmitting or receiving before entering Powerdown. The SFRs associated with the A/D and the serial port may also contain incorrect information when returning from Powerdown.

When the chip is in Powerdown, it is impossible for the watchdog timer to time out because its clock has stopped. Systems which must use the Watchdog and Powerdown, should clear the Watchdog right before entering Powerdown. This will keep the Watchdog from timing out when the oscillator is stabilizing after leaving Powerdown. Also, the Clock Detect Enable circuit should not be activated if the chip will enter the Powerdown Mode. For this reason the I_{OL} and I_{OH} in Reset Specifications must be carefully verified.

14.3 ONCE™ and Test Modes

Test Modes can be entered on the 80C196KB by holding ALE, INST or $\overline{RD}$ in their active state on the rising edge of $\overline{RESET}$. The only Test Mode not reserved for use by Intel is the ONCE, or ON-Circuit-Emulation Mode.

ONCE is entered by driving ALE and INST high and $\overline{RD}$ low on the rising edge of $\overline{RESET}$. All pins except XTAL1 and XTAL2 are floated. Some of the pins are not truly high impedance as they have weak pullups or pulldowns. The ONCE Mode is useful in electrically removing the 80C196KB from the rest of the system. A typical application of the ONCE Mode would be to program discrete EPROMs onboard without removing the 80C196KB from its socket.

ALE, INST, and $\overline{RD}$ are weakly pulled high or low during reset. It is important that a circuit does not inadvertantly drive these signals during reset, or a Test Mode could be entered by accident.

15.0 MEMORY CONTROLLER AND THE MEMORY MAP

The addressable memory space on the 80C196KB consists of 64K bytes. Locations that have special purposes are 0-0FFH and 1FFEH-2080H. All other locations are available as program or data storage or for memory mapped peripherals. Figure 15-1 shows the 80C196KB memory map.

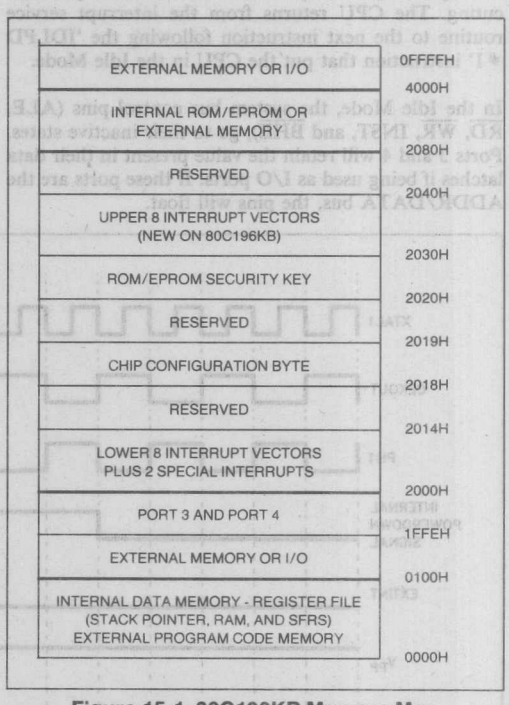


Figure 15-1. 80C196KB Memory Map

15.1 Memory Controller

All of the program memory and external data memory are transferred to the CPU through the memory controller. The memory controller consists of a Slave Program Counter, an Instruction Queue, and a bus controller.

The Slave Program Counter keeps track of the instructions fetched from program memory. Instructions fetched by the slave program counter are stored in the queue. The Slave Program Counter may be up to four bytes ahead of the Program Counter because it is prefetching instructions.

When debugging code using a logic analyzer, a designer must be aware of the queue. It is not possible to determine when an instruction will execute simply by tracing the external bus. The queue is filled in advance of executing an instruction.

The bus controller accesses program memory and external data memory and arbitrates between instruction fetches and data reads and writes. The bus controller supports both 8-bit and 16-bit external bus modes. Different modes of the bus controller and external memory interface are discussed in detail in the Section 6.

15.2 Memory Map and Reserved Locations

Locations 00H-0FFH contain the Register File and Special Function Registers (SFRs). No code can be executed from these locations. Code fetches from this area will be directed to external memory. 00H-0FFH external memory locations are reserved for use by Intel development tools and are not available to the user.

The RALU (Register Arithmetic Logic Unit) can operate directly on any of the 256 internal register locations. Locations 18H and 19H contain the stack pointer. The stack pointer can be used as standard RAM if stack operations are not being performed. The stack pointer must be initialized by software and can point anywhere in memory space. There are no restrictions on the remaining 230 bytes of registers except that code can not be executed from them. For more information on the RALU and its functions, refer to Section 2.

Locations 00H through 17H are the Special Function Registers, or SFRs. The SFRs control the onboard peripherals of the 80C196KB. For more detailed information on a particular SFR, refer to the peripherals section.

The 8K of internal memory available on the ROM and EPROM versions of the 80C196KB reside in locations 2000H-3FFFH. If $\overline{EA}$ is a TTL high, fetches from these locations will be directed internally. All reserved locations in internal memory must be programmed to 0FFH for future compatibility.

RESERVED MEMORY SPACES

The reserved locations must be filled with 0FFHs to remain compatible with future products and are shown in Figure 15-2. Any other values may cause unpredictable results.

Locations 1FFEH and 1FFFH are reserved for the data latches of Ports 3 and 4. More information is given on the functions of these ports in the peripherals section. The lower 10 interrupt vectors are stored in locations 2000H-2013H. Location 2014H contains the PPW (Programming Pulse Width) register. The PPW register is used solely to program 87C196KB EPROM parts and is a reserved location on ROM and ROMLESS versions. 2015H-2017H are reserved locations.

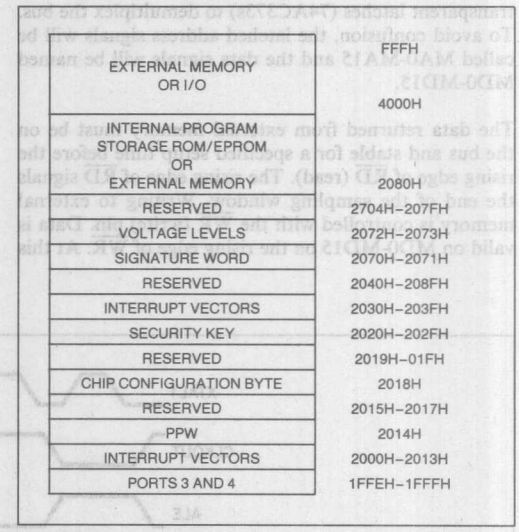


Figure 15-2. Reserved Memory Locations

The Chip Configuration Byte is stored at location 2018H. Locations 2019H-201FH are reserved by Intel. 2020H-202FH contains the ROM/EPROM security key. The most significant 8 interrupt vectors are at locations 2030H-203EH. Locations 2040-206FH are also reserved by Intel. Locations 2070H-2073H contain the signature word. The signature word contains programming voltage information primarily for programmer manufacturers to determine which programming algorithm to use. Finally, locations 2074H-207FH are reserved by Intel.

16.0 EXTERNAL MEMORY INTERFACING

16.1 Bus Operation

There are several different external operating modes on the 80C196KB. The standard bus mode uses a 16 bit multiplexed address/data bus. Other bus modes include an 8 bit external bus mode and a mode in which the bus size can be dynamically switched between 8-bits and 16-bits. In addition, there are several options available on the type of bus control signals which make an external bus simple to design.

In the standard mode, external memory is addressed through lines AD0-AD15 which form a 16 bit multiplexed bus. The address/data bus shares pins with ports 3 and 4. Figure 16-1 shows an idealized timing diagram for the external bus signals.

Address Latch Enable (ALE) provides a strobe to transparent latches (74AC373s) to demultiplex the bus. To avoid confusion, the latched address signals will be called MA0-MA15 and the data signals will be named MD0-MD15.

The data returned from external memory must be on the bus and stable for a specified setup time before the rising edge of $\overline{RD}$ (read). The rising edge of $\overline{RD}$ signals the end of the sampling window. Writing to external memory is controlled with the $\overline{WR}$ (write) pin. Data is valid on MD0-MD15 on the rising edge of $\overline{WR}$. At this time data must be latched by the external system. The 80C196KB has ample setup and hold times for writes.

When $\overline{BHE}$ is asserted, the memory connected to the high byte of the data bus is selected. When MA0 is a 0, the memory connected to the low byte of the data bus is selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0 = 0, $\overline{BHE}$ = 1), to the high (odd) byte only (MA0 = 1, $\overline{BHE}$ = 0), or the both bytes (MA0 = 0, $\overline{BHE}$ = 0).

When a block of memory is decoded for reads only, the system does not have to decode $\overline{BHE}$ and MA0. The 80C196KB will discard the byte it does not need. For systems that write to external memory, a system must generate separate write strobes to both the high and low byte of memory. This is discussed in more detail later.

All of the external bus signals are gated by the rising and falling edges of CLKOUT. A zero waitstate bus cycle consists of two CLKOUT periods. Therefore, there are 4 clock edges that generate a complete bus cycle. The first falling edge of CLKOUT asserts ALE and drives an address on the bus. The rising edge of CLKOUT drives ALE inactive. The next falling edge of CLKOUT asserts $\overline{RD}$ (read) and floats the bus for a read cycle. During a $\overline{WR}$ (write) cycle, this edge asserts $\overline{WR}$ and drives valid data on the bus. On the last rising edge of CLKOUT, data is latched into the 80C196KB for a read cycle, or data is valid for a write cycle.

READY PIN

The READY pin can insert wait states into the bus cycle for interfacing to slow memory or peripherals. A

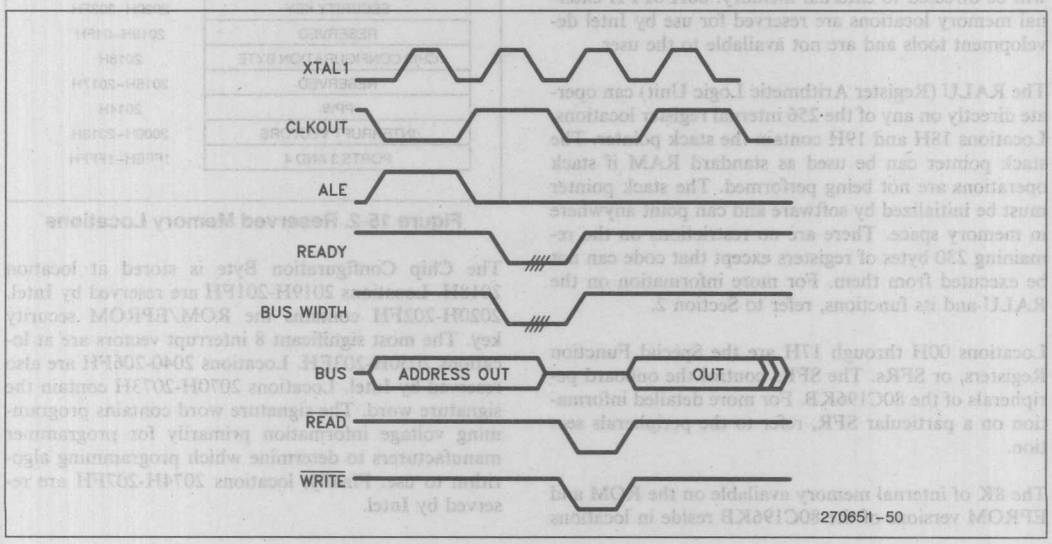


Figure 16-1. Idealized Bus Timings

wait state is 2 Tosc in length. Since the bus is synchronized to CLKOUT, it can only be held for an integral number of waitstates. Because the 80C196KB is a completely static part, the number of waitstates that can be inserted into a bus cycle is unbounded. Refer to the next section for information on internally controlling the number of waitstates inserted into a bus cycle.

There are several setup and hold times associated with the READY signal. If these timings are not met, the part may insert the incorrect number of waitstates.

INST pin

The INST pin is useful for decoding more than 64K of addressing space. The INST pin allows both 64K of code space and 64K of data space. For instruction fetches from external memory, the INST pin is asserted, or high for the entire bus cycle. For data reads and writes, the INST pin is low. The INST pin is low for the Chip Configuration Byte fetch and for interrupt vector fetches.

16.2 Chip Configuration Register

The CCR (Chip Configuration Register) is the first byte fetched from memory following a chip reset. The CCR is fetched from the CCB (Chip Configuration Byte) at location 2018H in either internal or external memory depending on the state of the $\overline{EA}$ pin. The CCR is only written once during the reset sequence. Once loaded, the CCR cannot be changed until the next reset.

The CCR is shown in Figure 16-2. The two most significant bits control the level of ROM/EPROM protection. ROM/EPROM protection is covered in the last section. The next two bits control the internal READY mode. The next three bits determine the bus control signals. The last bit enables or disables the Powerdown Mode.

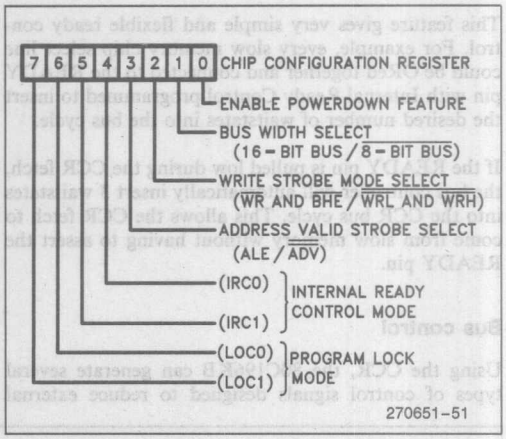


Figure 16-2. Chip Configuration Register

READY control

To simplify ready control, four modes of internal ready control are available. The modes are chosen by bits 4 and 5 of the CCR and are shown in Figure 16-3.

IRC1	IRC0	Description
0	0	Limit to one wait state
0	1	Limit to two wait states
1	0	Limit to three wait states
1	1	Wait states not limited internally

Figure 16-3. Ready control modes

The internal ready control logic limits the number of waitstates that slow devices can insert into the bus cycle. When the READY pin is pulled low, waitstates are inserted into the bus cycle until the READY pin goes high, or the number of waitstate equal the number programmed into the CCR. So the ready control is a simple logical OR between the READY pin and the internal ready control.

This feature gives very simple and flexible ready control. For example, every slow memory chip select line could be ORed together and connected to the READY pin with Internal Ready Control programmed to insert the desired number of waitstates into the bus cycle.

If the READY pin is pulled low during the CCR fetch, the bus controller will automatically insert 3 waitstates into the CCR bus cycle. This allows the CCR fetch to come from slow memory without having to assert the READY pin.

Bus control

Using the CCR, the 80C196KB can generate several types of control signals designed to reduce external hardware. The ALE, $\overline{WR}$, and $\overline{BHE}$ pins serve dual functions. Bits 2 and 3 of the CCR specify the function performed by these control lines.

Standard bus control

If CCR bits 2 and 3 are 1s, the standard bus control signals ALE, $\overline{WR}$, and $\overline{BHE}$ are generated as shown in Figure 16-4. ALE rises as the address starts to be driven, and falls to externally latch the address. $\overline{WR}$ is driven for every write. $\overline{BHE}$ and MA0 can be combined to form $\overline{WRL}$ and $\overline{WRH}$ for even and odd byte writes.

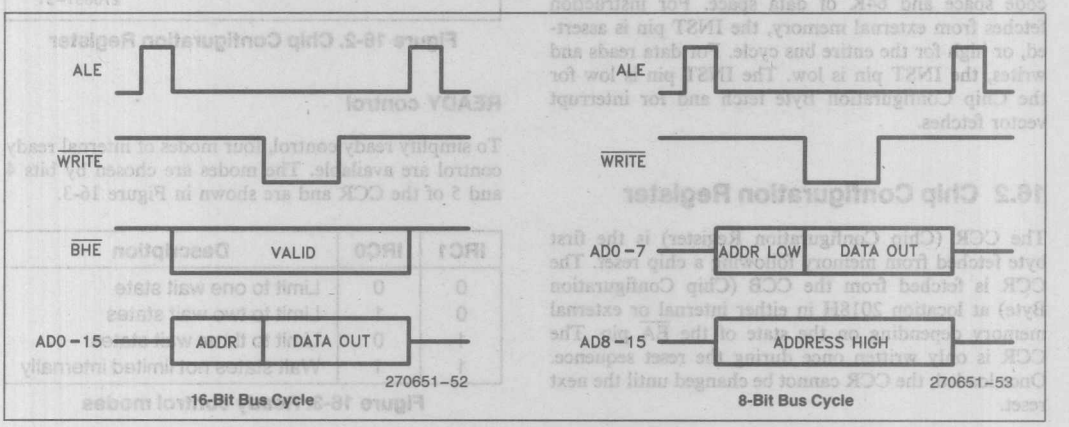


Figure 16-4. Standard Bus Control

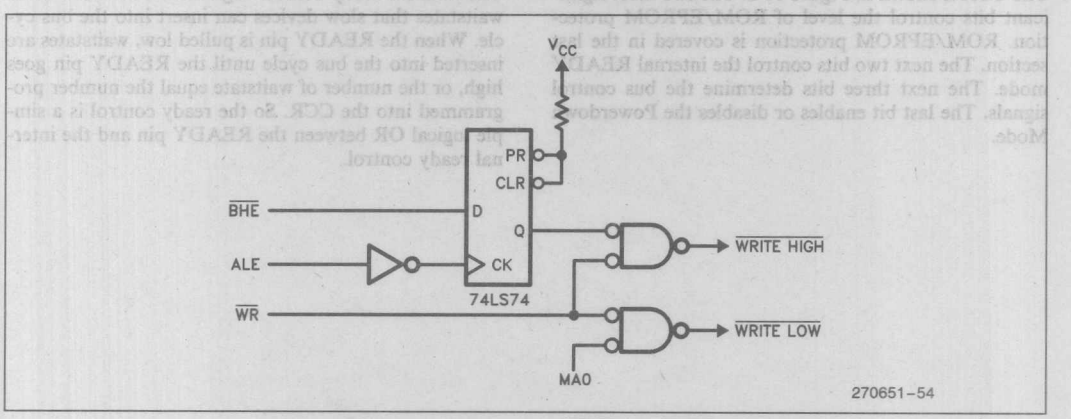


Figure 16-5. Decoding $\overline{WRL}$ and $\overline{WRH}$

Figure 16-5 is an example of external circuitry to decode $\overline{WRL}$ and $\overline{WRH}$.

Write Strobe Mode

The Write Strobe Mode eliminates the need to externally decode for odd and even byte writes. If CCR bit 2 is 0, and the bus is a 16-bit cycle, $\overline{WRL}$ and $\overline{WRH}$ are generated in place of $\overline{WR}$ and $\overline{BHE}$. $\overline{WRL}$ is asserted for all byte writes to an even address and all word writes. $\overline{WRH}$ is asserted for all byte writes to odd addresses and all word writes. The Write Strobe mode is shown in Figure 16-6.

In the eight bit mode, $\overline{WRL}$ and $\overline{WRH}$ are asserted for both even and odd addresses.

Address Valid Strobe Mode

Address Valid strobe replaces ALE if CCR bit 3 is 0. When Address valid Strobe mode is selected, $\overline{ADV}$ will be asserted after an external address is setup. It will stay asserted until the end of the bus cycle as shown in Figure 16-7. $\overline{ADV}$ can be used as a simple chip select for external memory. $\overline{ADV}$ looks exactly like ALE for back to back bus cycles. The only difference is $\overline{ADV}$ will be inactive when the external bus is idle.

Address Valid with Write Strobe

If CCR bits 2 and 3 are 0, the Address Valid with Write Strobe mode is enabled. Figure 16-8 shows the signals.

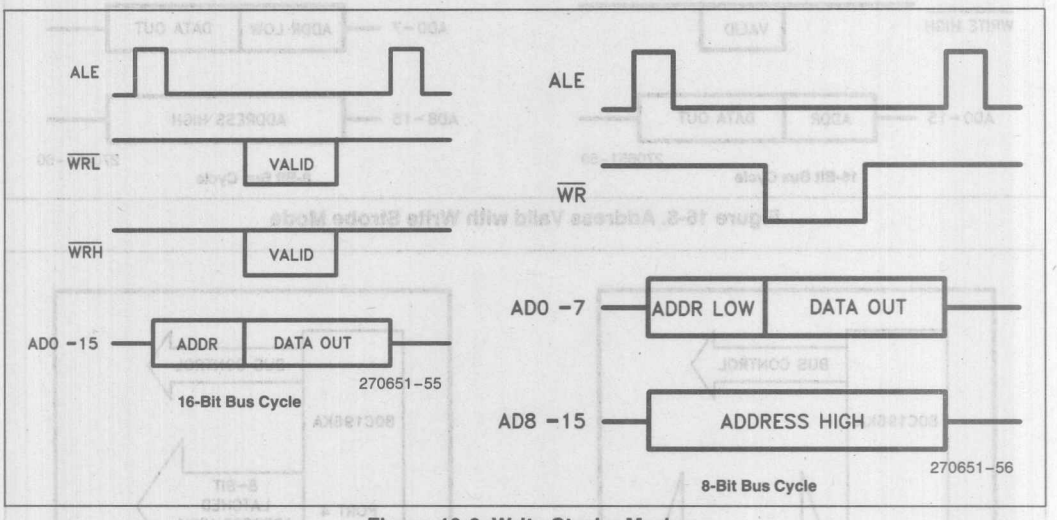


Figure 16-6. Write Strobe Mode

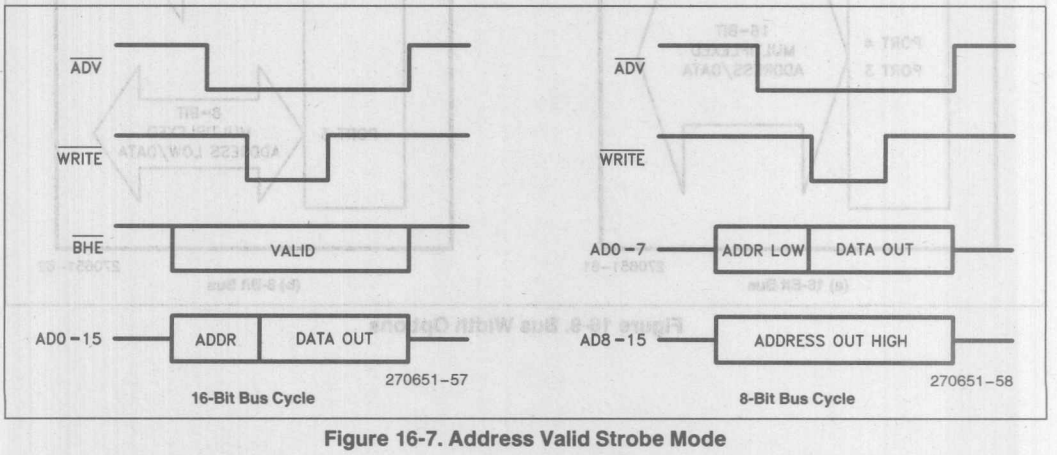


Figure 16-7. Address Valid Strobe Mode

16.3 Bus Width

The 80C196KB external bus width can be run-time conFigured to operate as a 16 bit multiplexed address/data bus, or as an MCS-51 style multiplexed 16 bit address/8 bit data bus.

During 16 bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed with address/data but Port 4 only outputs the upper 8 address bits. The Addresses on Port 4 are valid throughout the entire bus cycle. Figure 16-9 shows the two bus width options.

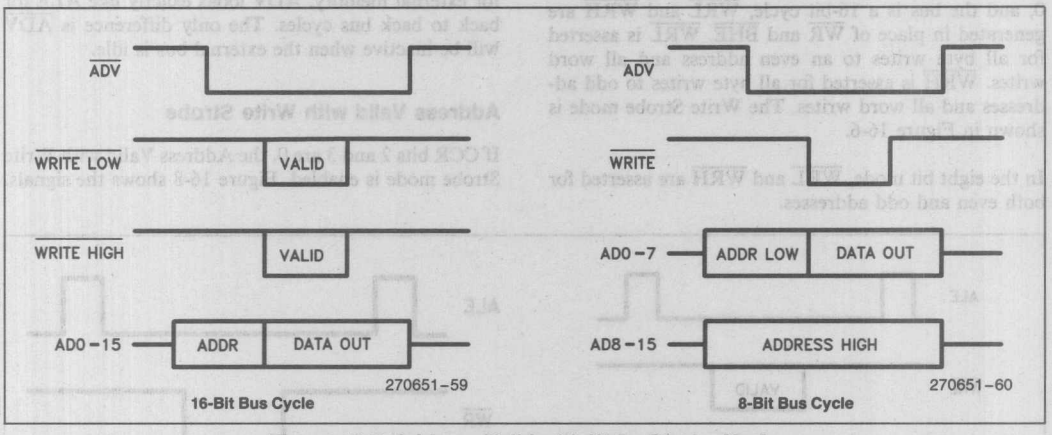


Figure 16-8. Address Valid with Write Strobe Mode

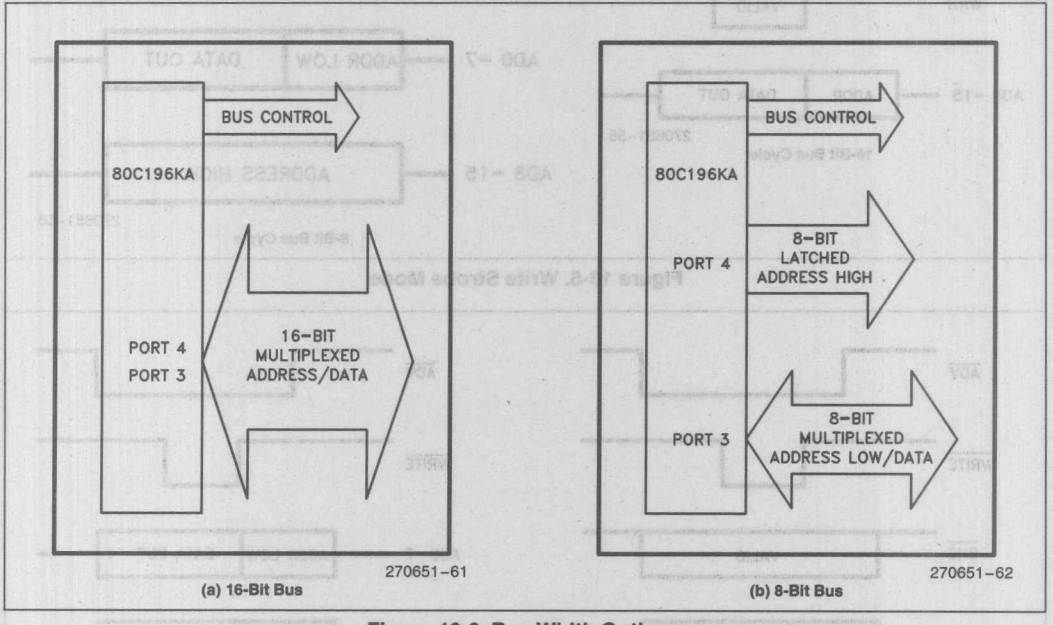


Figure 16-9. Bus Width Options

The external bus width can be changed every bus cycle if a 1 was loaded into bit CCR.1 at reset. The bus width is changed on the fly by using the BUSWIDTH pin. If the BUSWIDTH pin is a 1, the bus cycle is 16-bits. For an 8-bit bus cycle, the BUSWIDTH pin is a zero. The BUSWIDTH is sampled by the 80C196KB after the address is on the bus. The BUSWIDTH pin has about the same timing as the READY pin.

Applications for the BUSWIDTH pin are numerous. For example, a system could have code fetched from 16 bit memory, while data would come from 8 bit memory. This saves the cost of using two 8 bit static RAMS if only the capacity of one is needed. This system could be easily implemented by tying the chip select input of the 8-bit memory to the BUSWIDTH pin.

If CCR bit 1 is a 0, the 80C196KB is locked into the 8 bit mode and the BUSWIDTH pin is ignored.

When executing code from a 8-bit bus, some performance degradation is to be expected. The prefetch queue cannot be kept full under all conditions from an 8-bit bus. Also, word reads and writes to external memory will take an extra bus cycle for the extra byte.

16.4 $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ Protocol

The 80C196KB supports a bus exchange protocol, allowing other devices to gain control of the bus. The protocol consists of three signals, $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$, and $\overline{\text{BREQ}}$ which are multiplexed with 3 pins of port1. $\overline{\text{HOLD}}$ is an input asserted by a device which requests the 80C196KB bus. Figure 16-10 shows the timing for $\overline{\text{HOLD}}/\overline{\text{HLDA}}$. The 80C196KB responds by releasing the bus and asserting $\overline{\text{HLDA}}$. When the device is done accessing the 80C196KB memory, it relinquishes the bus by deactivating the $\overline{\text{HOLD}}$ pin. The 80C196KB will remove its $\overline{\text{HDLA}}$ and assume control of the bus. The third signal, $\overline{\text{BREQ}}$, is asserted by the 80C196KB during the hold sequence when it has a pending external bus cycle. The 80C196KB deactivates $\overline{\text{BREQ}}$ at the same time it deactivates $\overline{\text{HDLA}}$.

The $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$, and $\overline{\text{BREQ}}$ pin are multiplexed with P1.7, P1.6, and P1.5, respectively. To enable $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$, the HLDEN bit (WSR.7) must be to 1. HLDEN is cleared during reset. Once this bit is set, the port1 pins cannot be returned to being quasi-bidirectional pins. The $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ feature, however, can be disabled by clearing the HLDEN bit.

The $\overline{\text{HOLD}}$ and $\overline{\text{BREQ}}$ are sampled on phase 1, or when CLKOUT is low.

When the 80C196KB acknowledges the hold request, the output buffers for the addr/data bus, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{BHE}}$ and INST are floated. Although the strong pullup and pulldown on ALE are disabled, a weak pulldown is turned on to provide the option to wire OR ALE with other bus masters. The request to hold latency is dependent on the state of the bus controller.

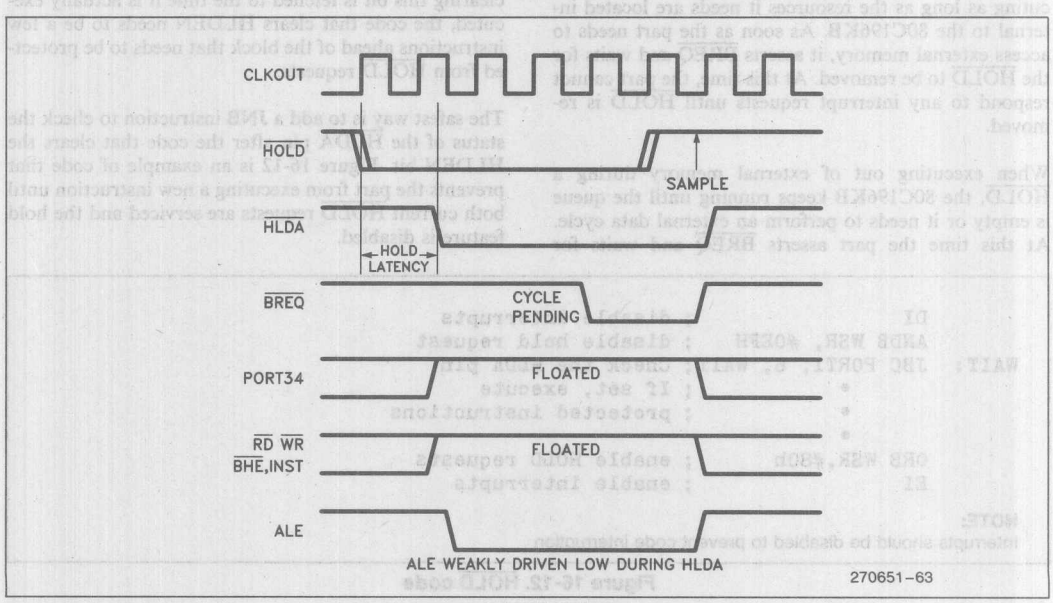


Figure 16-10. $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ Timings

MAXIMUM HOLD LATENCY

The maximum hold latency is defined as the maximum time before the 80C196KB will respond with HLDA to an incoming HOLD. The minimum latency is half a state time with one or two states added depending if the part is executing out of internal or external memory. When executing out of external memory, one state time is also added for each waitstate inserted into the current bus cycle. Figure 16-11 shows the maximum latency.

	Max Hold Latency
Internal Execution	1½ States
External Execution	2½ States

Figure 16-11. Maximum HOLD latency

REGAINING BUS CONTROL

There is no delay from the time the 80C196KB removes HLDA to the time it takes control of the bus. After HOLD is removed, the 80C196KB drops HLDA in the following state and resumes control of the bus.

BREQ is asserted when the part is in hold and needs to perform an external memory cycle. Once asserted, it remains asserted until HOLD is removed. At the earliest, BREQ can be asserted with HLDA.

Hold requests do not freeze the 80C196KB when executing out of internal memory. The part continues executing as long as the resources it needs are located internal to the 80C196KB. As soon as the part needs to access external memory, it asserts BREQ and waits for the HOLD to be removed. At this time, the part cannot respond to any interrupt requests until HOLD is removed.

When executing out of external memory during a HOLD, the 80C196KB keeps running until the queue is empty or it needs to perform an external data cycle. At this time the part asserts BREQ and waits for HOLD to be removed. The 80C196KB cannot service any interrupts until HOLD is removed.

The 80C196KB will also respond to hold requests in the Idle Mode. The latency for entering bus hold from the Idle Mode is the same as when executing out of internal memory.

Special consideration must be given to the bus arbiter design if the 80C196KB can be reset while in HOLD. For example, a CPU part would try and fetch the CCR from external memory after RESET is brought high. Now there would be two parts attempting to access 80C196KB memory. Also, if another bus master is directly driving ALE, RD, and INST, the ONCE mode or another test mode could be entered. The simplest solution is to make the RESET pin of the 80C196KB a system reset. This way the other bus master would also be reset. Examples of system reset circuits are given in Section 13.

DISABLING HOLD REQUESTS

Clearing the HLDEN bit (WSR.7), can disable HOLD requests when consecutive memory cycles are required. Clearing the HDLEN bit, however, does not cause the 80C196KB to take over the bus immediately. The 80C196KB waits for the current HOLD request to finish. Then it disables the bus hold feature, causing any new requests to be ignored until the HLDEN bit is set again. Since there is a delay from the time the code for clearing this bit is fetched to the time it is actually executed, the code that clears HLDEN needs to be a few instructions ahead of the block that needs to be protected from HOLD requests.

The safest way is to add a JNB instruction to check the status of the HLDA pin after the code that clears the HLDEN bit. Figure 16-12 is an example of code that prevents the part from executing a new instruction until both current HOLD requests are serviced and the hold feature is disabled.

```
        DI                  ; disable interrupts
        ANDB WSR, #0EFH     ; disable hold request
WAIT:   JBC PORT1, 6, WAIT; Check the HLDA pin
              •             ; If set, execute
              •             ; protected instructions
              •
        ORB WSR,#80h        ; enable HOLD requests
        EI                  ; enable interrupts
```

NOTE:
Interrupts should be disabled to prevent code interruption

Figure 16-12. HOLD code

16.5 AC Timing Explanations

Figure 16-13 shows the timing of the ADDR/DATA bus and control signals. Refer to the latest data sheet for the AC timings to make sure your system meets specifications. The major timing specifications are explained in English in Figure 16-14.

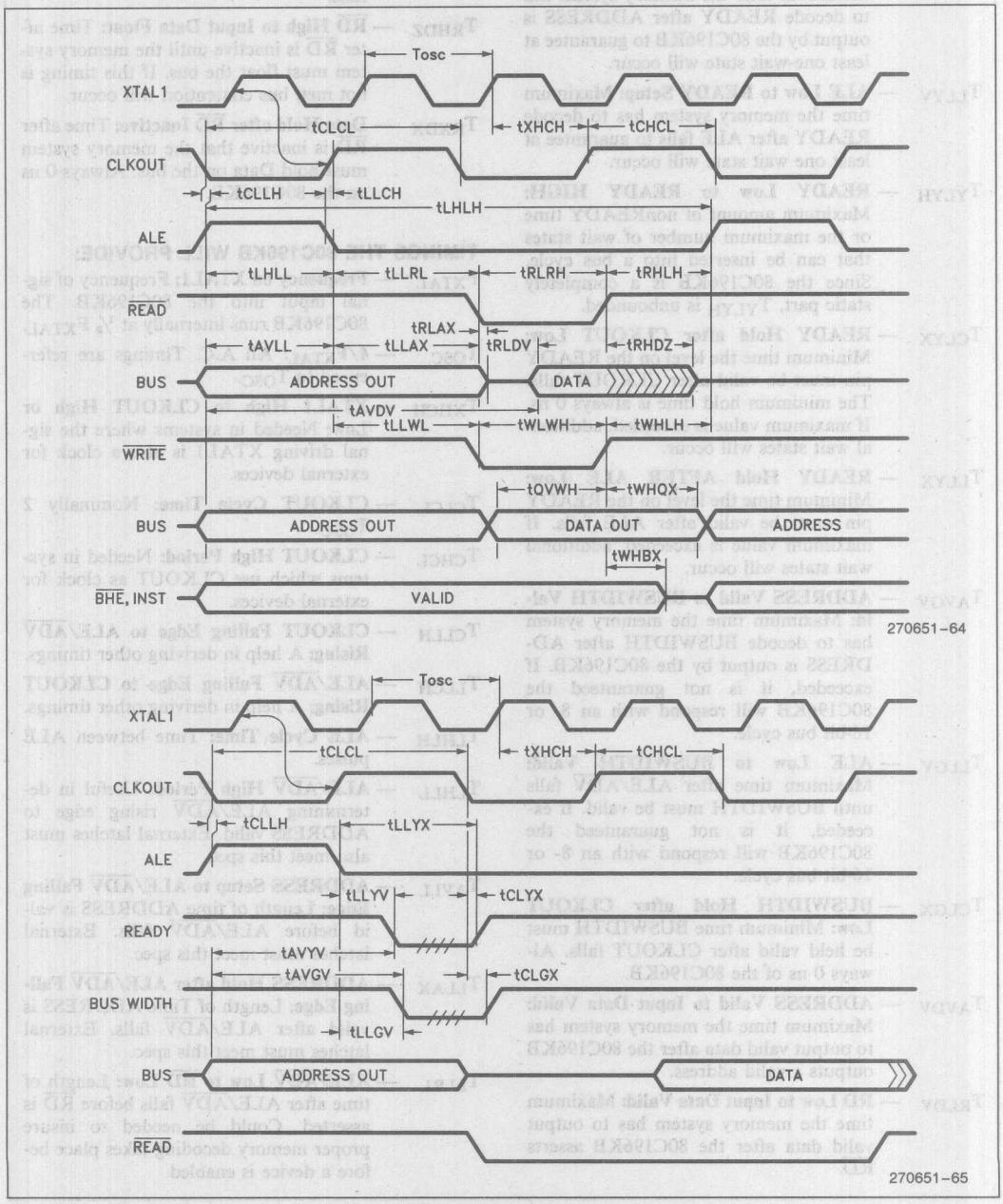


Figure 16-13. AC Timing Diagrams

TIMINGS THE MEMORY SYSTEM MUST MEET:

T_{AVYV} — **ADDRESS Valid to READY Setup:** Maximum time the memory system has to decode READY after ADDRESS is output by the 80C196KB to guarantee at least one-wait state will occur.

T_{LLYV} — **ALE Low to READY Setup:** Maximum time the memory system has to decode READY after ALE falls to guarantee at least one wait state will occur.

T_{YLYH} — **READY Low to READY HIGH:** Maximum amount of nonREADY time or the maximum number of wait states that can be inserted into a bus cycle. Since the 80C196KB is a completely static part, T_{YLYH} is unbounded.

T_{CLYX} — **READY Hold after CLKOUT Low:** Minimum time the level on the READY pin must be valid after CLKOUT falls. The minimum hold time is always 0 ns. If maximum value is exceeded, additional wait states will occur.

T_{LLYX} — **READY Hold AFTER ALE Low:** Minimum time the level on the READY pin must be valid after ALE falls. If maximum value is exceeded, additional wait states will occur.

T_{AVGV} — **ADDRESS Valid to BUSWIDTH Valid:** Maximum time the memory system has to decode BUSWIDTH after ADDRESS is output by the 80C196KB. If exceeded, it is not guaranteed the 80C196KB will respond with an 8- or 16-bit bus cycle.

T_{LLGV} — **ALE Low to BUSWIDTH Valid:** Maximum time after ALE/$\overline{\text{ADV}}$ falls until BUSWIDTH must be valid. If exceeded, it is not guaranteed the 80C196KB will respond with an 8- or 16-bit bus cycle.

T_{CLGX} — **BUSWIDTH Hold after CLKOUT Low:** Minimum time BUSWIDTH must be held valid after CLKOUT falls. Always 0 ns of the 80C196KB.

T_{AVDV} — **ADDRESS Valid to Input Data Valid:** Maximum time the memory system has to output valid data after the 80C196KB outputs a valid address.

T_{RLDV} — **$\overline{\text{RD}}$ Low to Input Data Valid:** Maximum time the memory system has to output valid data after the 80C196KB asserts $\overline{\text{RD}}$.

T_{CLDV} — **CLKOUT Low to Input Data Valid:** Maximum time the memory system has to output valid data after the CLKOUT falls.

T_{RHDZ} — **$\overline{\text{RD}}$ High to Input Data Float:** Time after $\overline{\text{RD}}$ is inactive until the memory system must float the bus. If this timing is not met, bus contention will occur.

T_{RXDX} — **Data Hold after $\overline{\text{RD}}$ Inactive:** Time after $\overline{\text{RD}}$ is inactive that the memory system must hold Data on the bus. Always 0 ns on the 80C196KB.

TIMINGS THE 80C196KB WILL PROVIDE:

F_{XTAL} — **Frequency on XTAL1:** Frequency of signal input into the 80C196KB. The 80C196KB runs internally at ½ F_{XTAL}.

T_{OSC} — **1/F_{XTAL}:** All A.C. Timings are referenced to T_{OSC}.

T_{XHCH} — **XTAL1 High to CLKOUT High or Low:** Needed in systems where the signal driving XTAL1 is also a clock for external devices.

T_{CLCL} — **CLKOUT Cycle Time:** Nominally 2 T_{OSC}.

T_{CHCL} — **CLKOUT High Period:** Needed in systems which use CLKOUT as clock for external devices.

T_{CLLH} — **CLKOUT Falling Edge to ALE/$\overline{\text{ADV}}$ Rising:** A help in deriving other timings.

T_{LLCH} — **ALE/$\overline{\text{ADV}}$ Falling Edge to CLKOUT Rising:** A help in deriving other timings.

T_{LHLH} — **ALE Cycle Time:** Time between ALE pulses.

T_{LHLL} — **ALE/$\overline{\text{ADV}}$ High Period:** Useful in determining ALE/$\overline{\text{ADV}}$ rising edge to ADDRESS valid. External latches must also meet this spec.

T_{AVLL} — **ADDRESS Setup to ALE/$\overline{\text{ADV}}$ Falling Edge:** Length of time ADDRESS is valid before ALE/$\overline{\text{ADV}}$ falls. External latches must meet this spec.

T_{LLAX} — **ADDRESS Hold after ALE/$\overline{\text{ADV}}$ Falling Edge:** Length of Time ADDRESS is valid after ALE/$\overline{\text{ADV}}$ falls. External latches must meet this spec.

T_{LLRL} — **ALE/$\overline{\text{ADV}}$ Low to $\overline{\text{RD}}$ Low:** Length of time after ALE/$\overline{\text{ADV}}$ falls before $\overline{\text{RD}}$ is asserted. Could be needed to insure proper memory decoding takes place before a device is enabled.

Figure 16-14. AC Timing Explanations

T_{RLCL} — **$\overline{RD}$ Low to CLKOUT Falling Edge:** Length of time from $\overline{RD}$ asserted to CLKOUT falling edge: Useful for systems based on CLKOUT.

T_{RLRH} — **$\overline{RD}$ Low to $\overline{RD}$ High:** $\overline{RD}$ pulse width.

T_{RHLH} — **$\overline{RD}$ High to ALE/$\overline{ADV}$ Asserted:** Time between $\overline{RD}$ going inactive and next ALE/$\overline{ADV}$, also used to calculate time between inactive and next ADDRESS valid.

T_{RLAX} — **$\overline{RD}$ Low to ADDRESS Float:** Used to calculate when the 80C196KB stops driving ADDRESS on the bus.

T_{LLWL} — **ALE/$\overline{ADV}$ Low Edge to $\overline{WR}$ Low:** Length of time ALE/$\overline{ADV}$ falls before $\overline{WR}$ is asserted. Could be needed to ensure proper memory decoding takes place before a device is enabled.

T_{CLWL} — **CLKOUT Falling Edge to $\overline{WR}$ Low:** Time between CLKOUT going low and $\overline{WR}$ being asserted. Useful in systems based on CLKOUT.

T_{QVWH} — **Data Valid to $\overline{WR}$ Rising Edge:** Time between data being valid on the bus and $\overline{WR}$ going inactive. Memory devices must meet this spec.

T_{CHWH} — **CLKOUT High to $\overline{WR}$ Rising Edge:** Time between CLKOUT going high and $\overline{WR}$ going inactive. Useful in systems based on CLKOUT.

T_{WLWH} — **$\overline{WR}$ Low to $\overline{WR}$ High:** $\overline{WR}$ pulse width. Memory devices must meet this spec.

T_{WHQX} — **Data Hold after $\overline{WR}$ Rising Edge:** Amount of time data is valid on the bus after $\overline{WR}$ going inactive. Memory devices must meet this spec.

T_{WHLH} — **$\overline{WR}$ Rising Edge to ALE/$\overline{ADV}$ Rising Edge:** Time between $\overline{WR}$ going inactive and next ALE/$\overline{ADV}$. Also used to calculate $\overline{WR}$ inactive and next ADDRESS valid.

T_{WHBX} — **$\overline{BHE}$, INST, Hold after $\overline{WR}$ Rising Edge:** Minimum time these signals will be valid after $\overline{WR}$ inactive.

Figure 16-14. AC Timing Explanations (Continued)

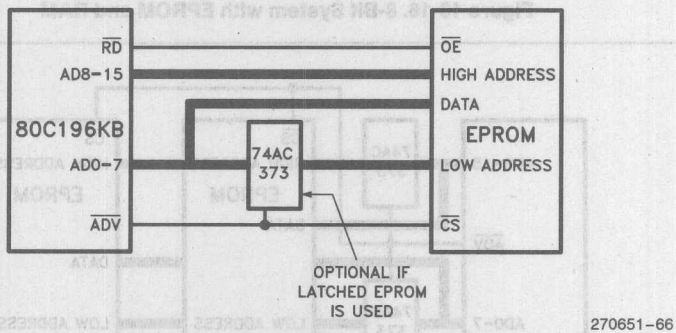


Figure 16-15. 8-Bit System with EPROM

16.6 Memory System Examples

External memory systems for the 80C196KB can be set up in many different ways. Figure 16-15 shows a simple 8 bit system with a single EPROM. The $\overline{ADV}$ Mode can be selected to provide a chip select to the memory. By setting bit CCR.1 to 0, the system is locked into the eight bit mode. An eight bit system with EPROM and RAM is shown in Figure 16-16. The EPROM is decoded in the lower half of memory,and the RAM in the upper half.

Figure 16-17 shows a 16 bit system with 2 EPROMs. Again, $\overline{ADV}$ is used to chip select the memory. Figure 16-18 shows a system with dynamic bus width. Code is executed from the two EPROMs and data is stored in the single RAM. Note the Chip Select of the RAM also is input to the BUSWIDTH pin to select an eight bit cycle.

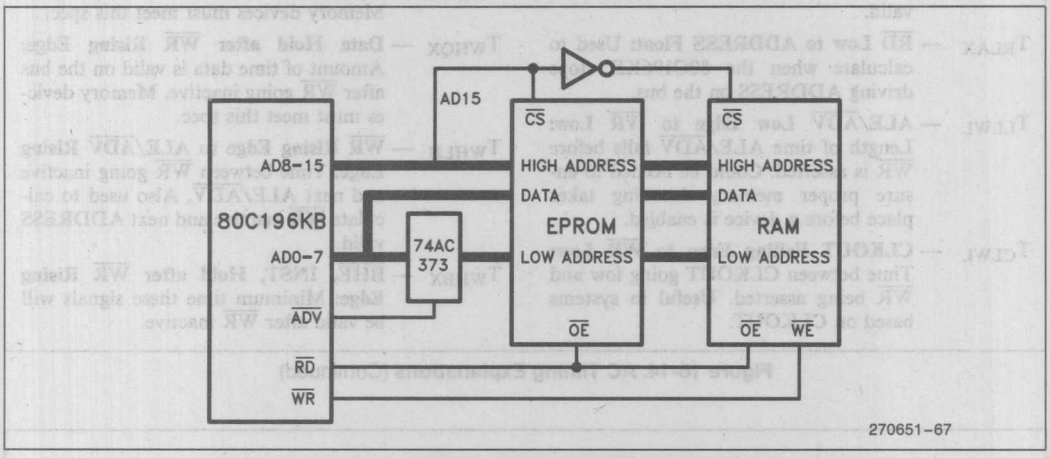


Figure 16-16. 8-Bit System with EPROM and RAM

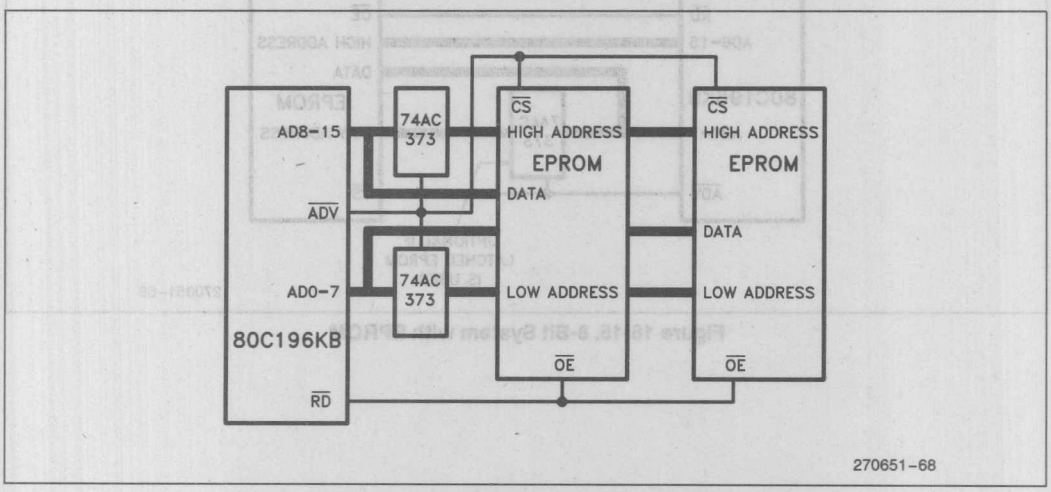


Figure 16-17. 16-Bit System with EPROM

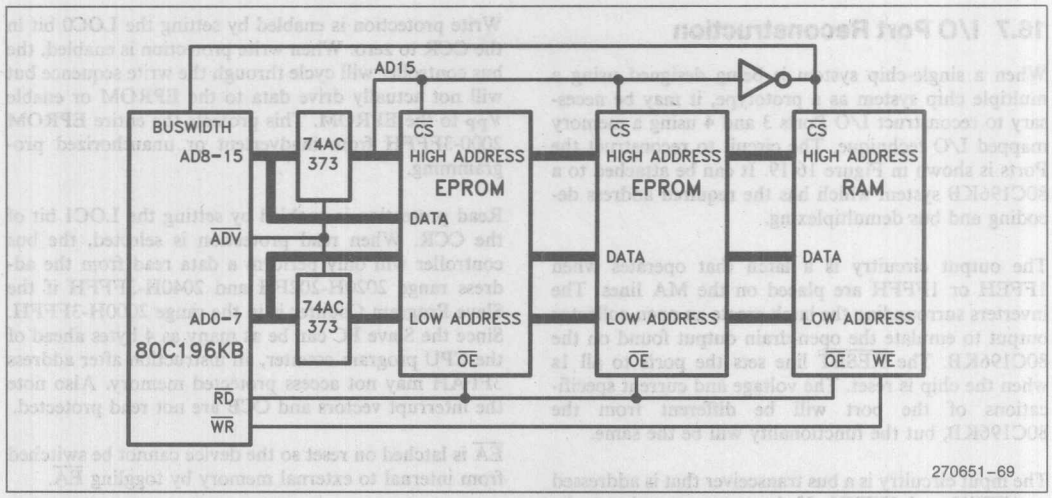


Figure 16-18. 16-Bit System with Dynamic Buswidth

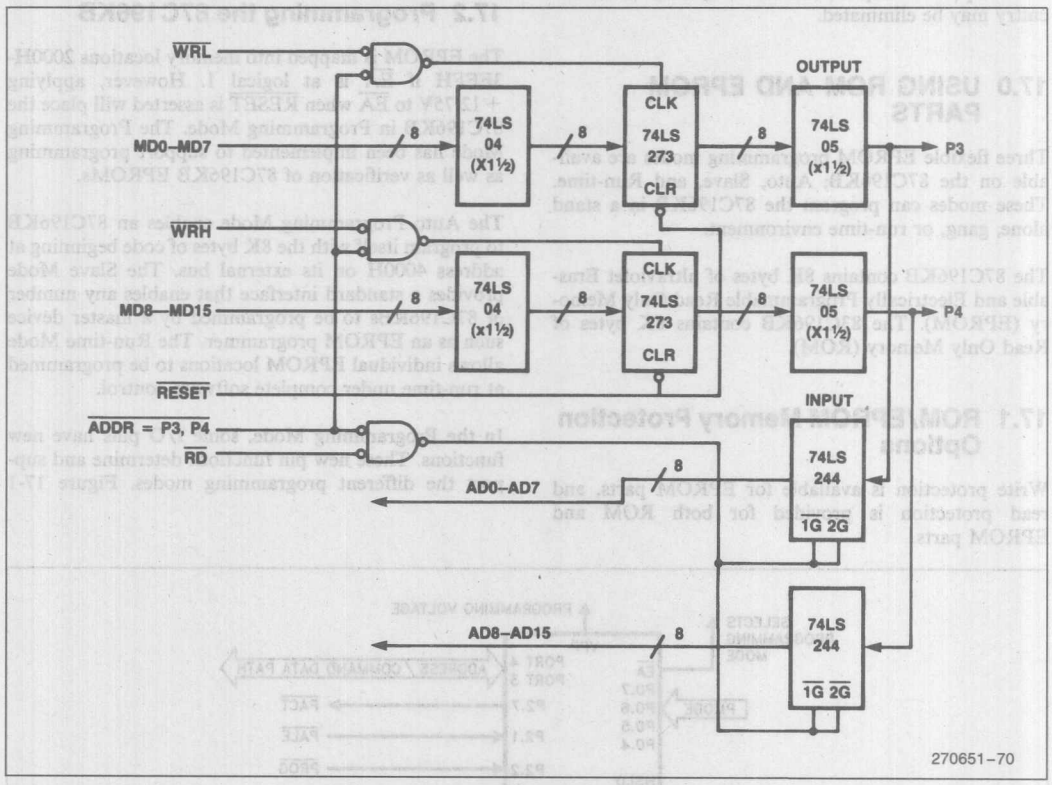


Figure 16-19. I/O Port Reconstruction

16.7 I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O Ports 3 and 4 using a memory mapped I/O technique. The circuit to reconstruct the Ports is shown in Figure 16-19. It can be attached to a 80C196KB system which has the required address decoding and bus demultiplexing.

The output circuitry is a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 80C196KB. The $\overline{\text{RESET}}$ line sets the ports to all 1s when the chip is reset. The voltage and current specifications of the port will be different from the 80C196KB, but the functionality will be the same.

The input circuitry is a bus transceiver that is addressed at 1FFEH and 1FFFH. If the ports are going to be either inputs or outputs, but not both, some of the circuitry may be eliminated.

17.0 USING ROM AND EPROM PARTS

Three flexible EPROM programming modes are available on the 87C196KB; Auto, Slave, and Run-time. These modes can program the 87C196KB in a stand alone, gang, or run-time environment.

The 87C196KB contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM). The 83C196KB contains 8K bytes of Read Only Memory (ROM).

17.1 ROM/EPROM Memory Protection Options

Write protection is available for EPROM parts, and read protection is provided for both ROM and EPROM parts.

Write protection is enabled by setting the LOC0 bit in the CCR to zero. When write protection is enabled, the bus controller will cycle through the write sequence but will not actually drive data to the EPROM or enable V_{PP} to the EPROM. This protects the entire EPROM 2000-3FFFH from inadvertent or unauthorized programming.

Read protection is enabled by setting the LOC1 bit of the CCR. When read protection is selected, the bus controller will only perform a data read from the address range 2020H-202FH and 2040H-3FFFH if the Slave Program Counter is in the range 2000H-3FFFH. Since the Slave PC can be as many as 4 bytes ahead of the CPU program counter, an instruction after address 3FFAH may not access protected memory. Also note the interrupt vectors and CCB are not read protected.

$\overline{\text{EA}}$ is latched on reset so the device cannot be switched from internal to external memory by toggling $\overline{\text{EA}}$.

17.2 Programming the 87C196KB

The EPROM is mapped into memory locations 2000H-3FFFH if $\overline{\text{EA}}$ is at logical 1. However, applying +12.75V to $\overline{\text{EA}}$ when $\overline{\text{RESET}}$ is asserted will place the 87C196KB in Programming Mode. The Programming Mode has been implemented to support programming as well as verification of 87C196KB EPROMs.

The Auto Programming Mode enables an 87C196KB to program itself with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 87C196KBs to be programmed by a master device such as an EPROM programmer. The Run-time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have new functions. These new pin functions determine and support the different programming modes. Figure 17-1

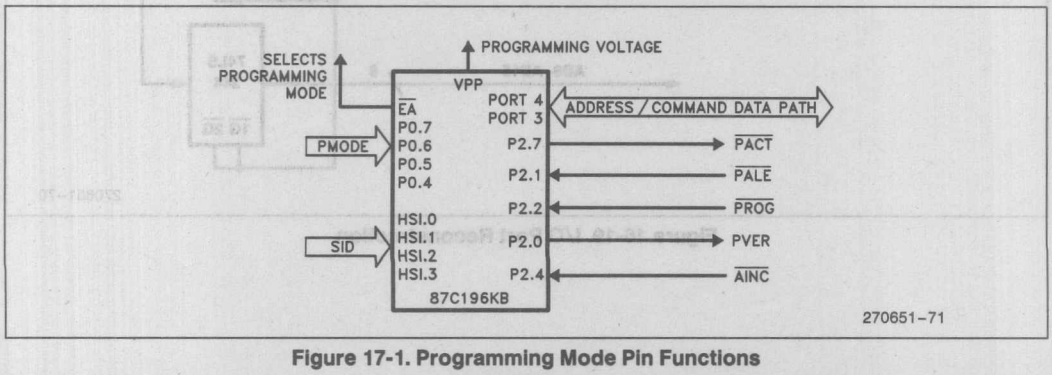


Figure 17-1. Programming Mode Pin Functions

Name	Function
PMODE	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 001 will use Port 3.1 to signal correct or incorrect program verification.
$\overline{\text{PALE}}$	Programming ALE Input. Accepted by an 87C196KB that is in Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming. Falling edge indicates valid data on PBUS and the beginning of programming. Rising edge indicates end of programming.
$\overline{\text{PACT}}$	Programming Active. Used in the Auto Programming Mode to indicate when programming activity is complete.
PVER	Program Verification. Signal is low after rising edge of PROG if the programming was not successful.
$\overline{\text{AINC}}$	Auto Increment. Active low signal indicates that the auto increment mode is enabled. Auto Increment will allow reading or writing of sequential EPROM locations without address transactions across the PBUS for each read or write.
PORTS	Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 87C196KBs. Used by chips in Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Should have pullups to V_{CC} (15 kΩ).

Figure 17-2. Programming Mode Pin Definitions

shows how the pins are renamed and Figure 17-2 describes in detail each new pin function.

While in Programming Mode, PMODE selects the programming function (see Figure 17-3). Run-time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 17-3. Programming Function Pmode Values

To guarantee proper functionality, the pins of PMODE and SID must be in their desired state before $\overline{\text{RESET}}$ rises. Once the part is reset, it should not be switched to another mode without a new reset sequence.

When $\overline{\text{EA}}$ selects the Programming Mode, the chip reset sequence loads the CCR from the PCCB (Programming Chip Configuration Byte). The PCCB is a separate EPROM location that is not mapped under normal operation. PCCB is important only when in the Auto Programming Mode. In this mode, the 87C196KB gets the programming data over the external bus. Therefore, the PCCB must correctly correspond to the memory system in the programming setup, which is not necessarily the memory system of the application.

The following sections describe the 87C196KB programming modes in detail.

PROGRAMMING PULSE WIDTH REGISTER (PPW)

When the 87C196KB programs itself the width of the programming pulse is determined by the 8 bit PPW (Programming Pulse Width) register. In the Auto Programming Mode, the PPW is loaded from location 4014H in external memory. In Run-time Programming Mode, the PPW is located in window 14 at 04H. In order for the EPROM to properly program, the pulse width must be set to approximately 100 uS. The pulse width is dependent on the oscillator frequency and is calculated with the following formula:

Pulse Width = PPW * (Tosc * 8)

PPW = 150 @ 12 Mhz

17.3 Auto Programming Mode

The Auto Programming Mode allows an 87C196KB EPROM to be programmed without a special EPROM programmer. In this mode, the 87C196KB simply programs itself with the data at external locations 4000H-5FFFH. Location 4014H in external memory contains the value that is loaded into the PPW. All that is required is that of external memory reside at these locations, $\overline{EA}$ is at 12.75V, the PPW is in location 4014H, and Vpp is applied. Figure 17-4 shows a minimum configuration using an 8K * 8 EPROM to program an 87C196KB in the Auto Programming Mode.

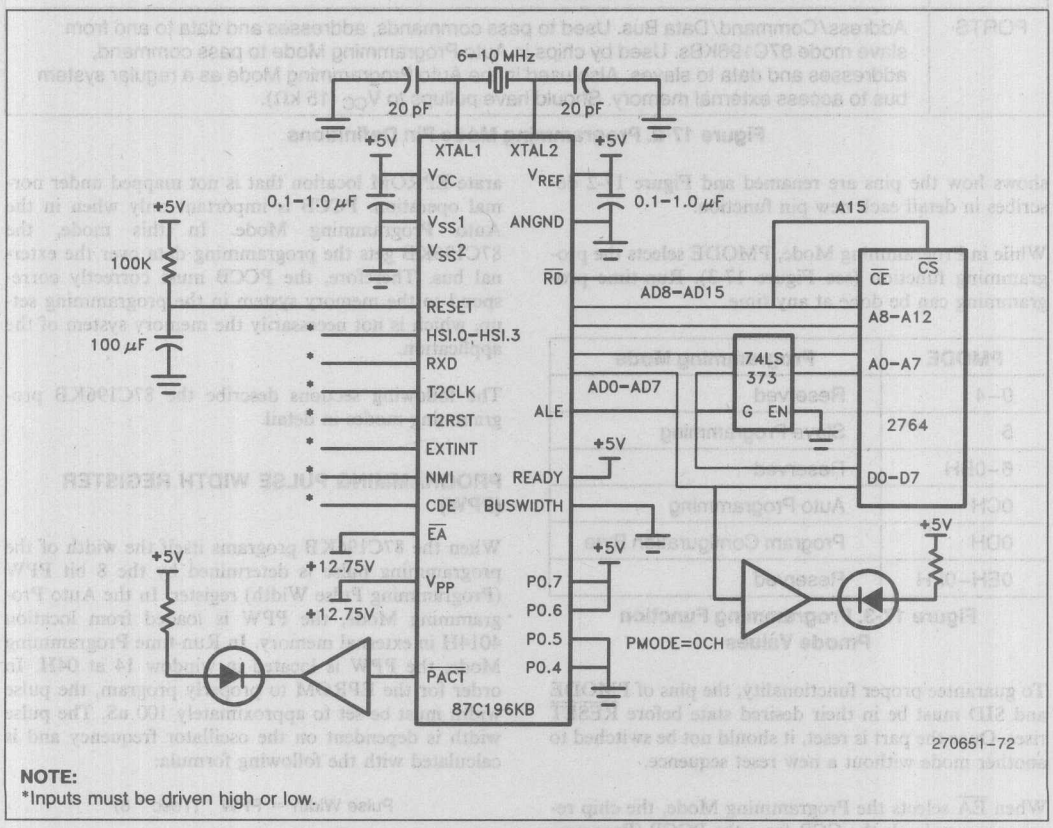
The 87C196KB first reads a word form external memory, then the Modified Quick-Pulse Programming Algorithm (described later) programs the corresponding EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations with 0FFH for data. When all 8K have been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on port 3.0 if it programmed correctly and a 1 if it failed.

Auto Configuration Byte Programming Mode

The CCB (2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But the Auto Configuration Byte Programming Mode allows a simple way to program the CCB when no other locations need be programmed. This mode also programs the PCCB. Figure 17-5 shows a block diagram for using the Auto Configuration Byte Programming Mode.

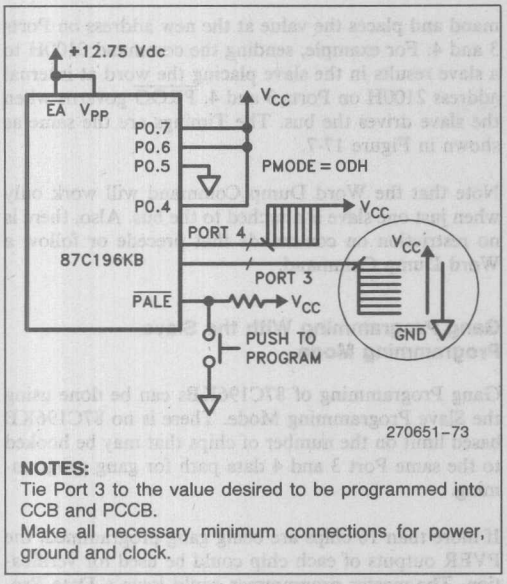
With PMODE = 0DH and 0FF on Port 4, the CCB and PCCB will be programmed with the data on Port 3 when a 0 is placed on PALE. After programming is complete, PVER is driven high if the bytes programmed correctly, and low if the programming failed.

This is the only way to program the PCCB. PCCB is a non-memory mapped location that gets loaded into the CCR when the 87C196KB is placed in Programming Mode. If the PCCB is not programmed, the CCR will be loaded with 0FFH when entering the Programming Mode.



NOTE:
*Inputs must be driven high or low.

Figure 17-4. Auto Programming Mode



NOTES:
Tie Port 3 to the value desired to be programmed into CCB and PCCB.
Make all necessary minimum connections for power, ground and clock.

Figure 17-5. The Auto CCB Programming Mode

17.4 Slave Programming Mode

Any number of 87C196KBs can be programmed by a master programmer through the Slave Programming Mode.

In this mode, the 87C196KB programs like a simple EPROM device. The 87C196KB responds to three different commands while in this mode: data program, data verify, and word dump. These commands, along with the transfer of appropriate data and addresses are selected using Ports 3 and 4 and five other pins for handshaking. The two most significant bits on Ports 3 and 4 specify the command and the lower 14 bits contain the address. The address ranges from 2000H-3FFFH and refers to internal memory space. Figure 17-6 is a list of valid Programming Commands.

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

Figure 17-6. Slave Programming Mode Commands

The 87C196KB receives an input signal, $\overline{\text{PALE}}$, to indicate a valid command is present. $\overline{\text{PROG}}$ causes the 87C196KB to read in or output a data word. An output signal, PVER, indicates if the programming was successful. $\overline{\text{AINC}}$ automatically increments the address for the Data Program and Word Dump commands. There is no 87C196KB dependent limit to the number of parts that can be gang programmed in the Slave Mode.

Data Program Command

A Data Program Command is illustrated in Figure 17-7. Asserting PALE latches the command and address on Ports 3 and 4. $\overline{\text{PROG}}$ is asserted to latch the data present on Ports 3 and 4. $\overline{\text{PROG}}$ also starts the actual programming sequence. The width of the $\overline{\text{PROG}}$ pulse determines the programming pulse width.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. PVER is asserted if the location programmed correctly. This gives verification information to programmers which can not use the Data Verify Command. The $\overline{\text{AINC}}$ pin can increment to the next location or a new Data Program Command can be issued.

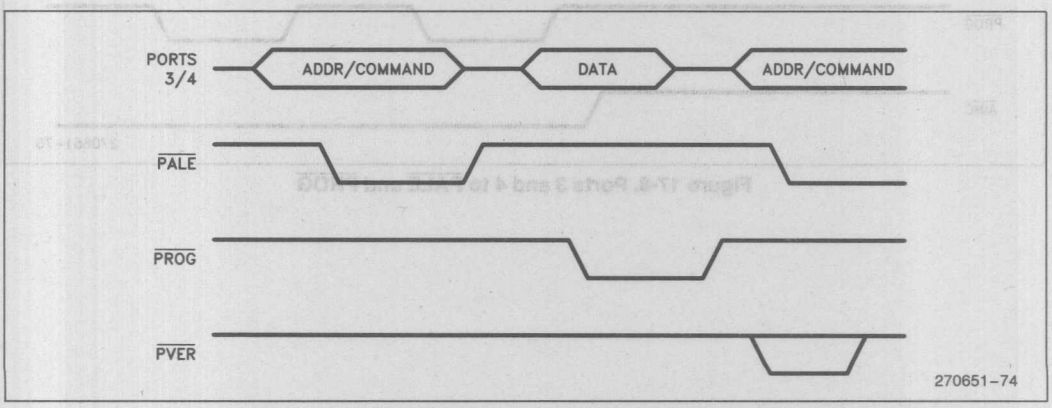


Figure 17-7. Data Program Command in Slave Mode

If PVER of all slaves are 1s after $\overline{PROG}$ rises then the data program was successful for every slave. If PVER is a 0 for any slave, then the part did not program correctly. Figure 17-7 shows the relationship of $\overline{PALE}$, $\overline{PROG}$, and PVER to the Command/Data path on Ports 3 and 4 for the Data Program Command.

Data Verify Command

When the Data Verify Command is sent, the slaves respond by driving one bit of Ports 3 and 4 to indicate correct or incorrect verification of the previous Data Program Command. A 1 indicates a correct verification, and a 0 indicates incorrect verification. The SID (Slave I.D) of each slave determines which bit of Ports 3 and 4 will be driven. For example, a SID of 0001 would drive Port 3.1. $\overline{PROG}$ governs when the slaves drive the bus. Figure 17-8 shows the relationship of ports 3 and 4 to $\overline{PALE}$ and $\overline{PROG}$.

A Data Verify Command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

Word Dump Command

When the Word Dump Command is issued, the 87C196KB adds 2000H to the address field of the command and places the value at the new address on Ports 3 and 4. For example, sending the command 0100H to a slave results in the slave placing the word at internal address 2100H on Ports 3 and 4. $\overline{PROG}$ governs when the slave drives the bus. The Timings are the same as shown in Figure 17-7.

Note that the Word Dump Command will work only when just one slave is attached to the bus. Also, there is no restriction on commands that precede or follow a Word Dump Command.

Gang Programming With the Slave Programming Mode.

Gang Programming of 87C196KBs can be done using the Slave Programming Mode. There is no 87C196KB based limit on the number of chips that may be hooked to the same Port 3 and 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER outputs of each chip could be used for verification. The master programmer could issue a Data Program Command, then either watch every chip's error signal, or AND all the signals together to form a system PVER.

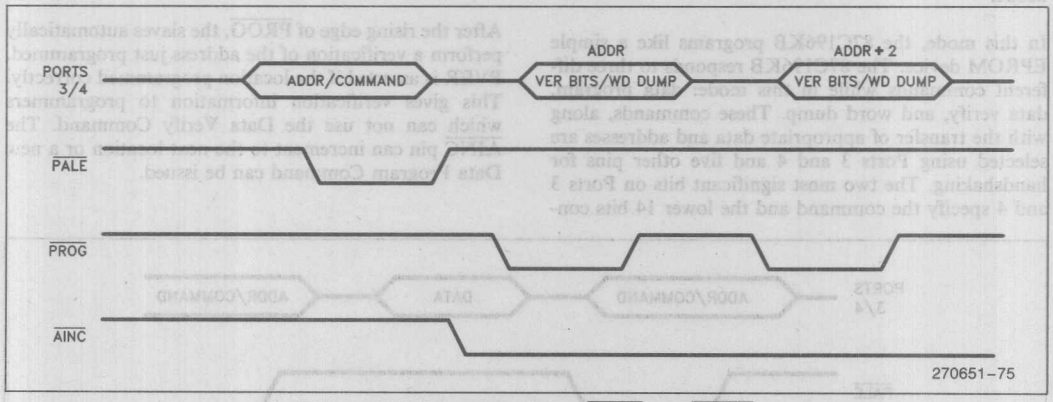


Figure 17-8. Ports 3 and 4 to $\overline{PALE}$ and $\overline{PROG}$

If 16 or fewer 87C196KBs are to be Gang programmed at once, a more flexible form of verification is available. By giving each chip a unique SID, the master programmer could issue a Data Verify Command after the Data Program Command. When a verify command is seen by the slaves, each will drive a bit of Ports 3 or 4 corresponding to its unique SID. A 1 driven signals the address verified, while a 0 means it failed.

Run-Time Programming

Run-Time Programming allows the user complete flexibility in the ways the internal EPROM is programmed. That flexibility includes the ability to program just one byte or word instead of the whole EPROM. The only additional requirement of a system doing Run-Time Programming is that programming voltage is applied V_{PP}. Run-Time Programming is done with $\overline{EA}$ at a TTL high.

To Run-Time Program, the user writes to the location to be programmed. The value of the PPW register determines the programming pulse. The PPW may not be available on future proliferations of the 80C196KB. To ensure future compatibility, the Idle Mode should be used for Run-Time Programming. Figure 17-9 is the recommended code sequence for Run-Time Programming. The Modified Quick Pulse algorithm guarantees the programmed EPROM cell for the life of the part.

A 'Security Key' mechanism and ROM Dump Mode have been implemented for testing internal ROM/EPROM.

The security key is a 128 bit number located in internal memory at locations 2020H-202FH The security key must be matched before a ROM dump will occur.

The ROM Dump Mode is entered just like any other Programming Mode (EA = 12.75V) with PMODE = 6H. The ROM Dump will write out the entire EPROM or ROM to locations 4000H-5FFFH in external memory.

The ROM Dump Mode begins with a security key verification. The user puts the same security key at external locations 4020H-402FH that is in internal locations 2020H-202FH. Before doing a ROM dump, the 87C196KB compares the two security keys. If they do not match, the 87C196KB enters an endless loop of internal execution.

When using the Auto Configuration byte or Auto Programming Modes, a security key verification is done if the CCB has read and or write protection enabled.

If the PCCB is programmed with any read or write protection, there is no way to enter any of the programming modes. So the last thing that should be done to

```
                LD WSR,#14                          ;Initialize programmable
                LD PPW,#VALUE                       ;pulse width

PROGRAM:        POP ADDRESS_TEMP                    ;Load program data
                POP DATA_TEMP                       ;and address
                PUSHF
                LD COUNT, #25T                      ;program using Modified
                                                     Quick Pulse
   LOOP:      * LDB INT_MASK,#ENABLE_SWT            ;program SWT for
              * LDB HSO_COMMAND,#SWTO_OVF           ;program pulse width
              * ADD HSO_TIME,TIMER1,#PROGRAM_PULSE
              * EI
                ST DATA_TEMP,[ADDR_TEMP]            ;enter idle mode until
              * IDLPD 1                             ;swt expires
                DJNZ COUNT, LOOP                    ;loop 25 times
                POPF
                RET

SWT_EXPIRED:     *POP 0                             ;service swt and return
              *RET
```

NOTE:
*Not Really Needed on Current 87C196KB Part

Figure 17-9. Future Run-Time Programming Algorithm

protect the part from unauthorized access, is to program the PCCB.

The Modified Quick Pulse Algorithm

The Modified Quick Pulse Algorithm must be used to guarantee programming over the life of the EPROM in Run-time and Slave Programming Modes.

The Modified Quick-Pulse Algorithm calls for each EPROM location to receive 25 separate 100 uS (±5 μs) programming cycles. Verification is done after the 25th pulse. If the location verifies, the next location is programmed. If the location fails to verify, the location fails the programming sequence.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 87C196KB EPROMs is done with V_{PP} = 12.75V ±0.25V and V_{CC} = 5.0V ±0.5V.

Signature Word

The 87C196KB contains a signature word at location 2070H. The word can be accessed in the Slave Mode by executing a Word Dump Command. The programming voltages are determined by reading the test ROM at locations 2072H and 2073H. The voltages are calculated by using the following equation.

Voltage = 20/256 * (test ROM data)

The values for the signature word and voltage levels are shown in Figure 17-10.

Description	Location	Value
Signature Word	2070H	897CH
Programming V_{CC}	2072H	040H (5.0V)
Programming V_{PP}	2073H	0A3H (12.75V)

Figure 17-10. Signature Word and Voltage Levels

Erasing the 87C196KB

After each erasure, all bits of the 87C196KB are logical '1s'. Data is introduced by selectively programming '0s'. The only way to change a '0' to a '1' is by exposure to ultraviolet light.

The erasure characteristics of the 87C196KB are so that erasure begins upon exposure to light with wavelengths shorter than approximately 4000 Angstroms. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000 Angstrom range. Constant exposure to room level fluorescent lighting could erase a 87C196KB in about 3 years. It would take about 1 week in direct sunlight to erase an 87C196KB.

Opaque labels should always be placed over the window to prevent unintentional erasure. In the Powerdown Mode, the part will draw more current than normal if the EPROM window is exposed to light.

The recommended erasure procedure for the 87C196KB is exposure to ultraviolet light which has a wavelength of 2537 Angstroms. The integrated dose (UV intensity * exposure time) should be a minimum of 15 Wsec/cm^2. The total time for erasure is about 15 to 20 minutes at this level of exposure. The 87C196KB should be placed within 1 inch of the lamp during exposure. The maximum integrated dose an 87C196KB can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 uW/cm^2). Exposure to UV light greater than this can cause permanent damage.

MCS®-96 Instruction Set

4

MCS®-96 Instruction Set

4

MCS®-96 INSTRUCTION SET

OVERVIEW

This chapter of the manual gives a description of each instruction recognized by the 8096. The instructions are sorted alphabetically by the mnemonic used in the assembly language for the 8096. A summary of the instruction set is included in Section 14 of the MCS®-96 Architecture chapter.

The instruction set descriptions in the following sections do not always show the effect on the program counter (PC). Unless otherwise specified, all instructions increment the PC by the number of bytes in the instruction.

A set of acronyms are used to make the instruction set descriptions easier to read, their definitions are listed below:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D".

baop. A byte operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D".

waop. A word operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

Lreg. A 32-bit register in the internal register file.

BEA. Extra bytes of code required for the address mode selected.

CEA. Extra state times (cycles) required for the address mode selected.

cadd. An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminant state after the operation.

Generic Jumps and Calls. The assembler for the MCS-96 family provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user's guide should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

Indirect Shifts. The indirect shift operations use registers 24 through 255 (18H–0FFH), since 0–15 are direct operators and registers 16 through 23 are Special Function Registers. Note that indirect shifts through SFRs are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 5 least significant bits.

1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

	DST	SRC
ADD	wreg,	waop

Object Code Format: [011001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

2. ADD (Three Operands) — ADD WORDS

Operation: The Sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADD	Dwreg,	Swreg,	waop

Object Code Format: [010001aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

	DST	SRC
ADDB	breg,	baop

Object Code Format: [011101aa] [baop] [breg]

Bytes 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADDB	Dbreg,	Sbreg,	baop

Object Code Format: [010101aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDC	wreg,	waop

Object Code Format: [101001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + BEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDCB	breg,	baop

Object Code Format: [101101aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
          DST     SRC
AND       wreg,   waop
```

Object Code Format: [011000aa] [waop] [wreg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
AND       Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
          DST     SRC
ANDB      breg,   baop
```

Object Code Format: [011100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
          DST     SRC1    SRC2
ANDB      Dbreg,  Sbreg,  baop
```

Object Code Format: [010100aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

11. BMOV — BLOCK MOVE (80C196KB only)

Operation: This instruction is used to move a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with auto-increment addressing modes. A long register addresses the source and destination pointers which are stored in adjacent word registers. The number of transfers is specified by a word register. The blocks of data can reside anywhere in memory, but should not overlap.

```
COUNT ← (CNTREG)
LOOP: SRCPTR ← (PTRS)
DSTPTR ← (PTRS + 2)
(DSTPTR) ← (SRCPTR)
(PTRS) ← SRCPTR + 2
(PTRS) + 2) ← DSTPTR + 2
COUNT ← COUNT − 1
if COUNT <> 0 then go to LOOP
```

PTRS CNTREG

Assembly Language Format: BMOV Lreg, wreg

Object Code Format: [11000001] [wreg] [Lreg]

Bytes: 3
States: internal/internal: 8 per transfer + 6
external/internal: 11 per transfer + 6
external/external: 14 per transfer + 6

Flags Affected							
Z	N	V	VT	C	X	I	ST
—	—	—	—	—	X	—	—

NOTES:
1. CNTREG does not get decremented during the instruction.
2. It is easy to unintentionally create a very long un-interruptable operation with this instruction.

To provide an interruptable version of the BMOV for large blocks, the BMOV instruction can be used with the DJNZ instruction. This is possible because the pointers are modified, but CNTREG is not. Consider the example:

```
      LD     PTRS, SRC          ;Pointer to base of sources table
      LD     PTRS+2, DST        ;Pointer to base of destination table
      LD     CNTREG, #COUNT     ;Number of bytes to move per set
      LD     CNTSET, #SETS      ;Number of sets to move
Move: BMOV   PTRS, CNTREG       ;Move one set
      DJNZ   CNTSET, MOVE       ;Decrement set counters and move again
```

12. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011] [wreg]

Bytes: 2
States: 8 (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

13. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001] [wreg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

14. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

15. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.

C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	0	—	—	—

16. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

17. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMP	wreg,	waop

Object Code Format: [100010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

18. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMPB	breg,	baop

Object Code Format: [100110aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

19. CMPL — COMPARE LONG (80C196KB only)

Operation: This instruction is used to compare the magnitudes of two double word (long) operands. The operands are specified using the direct addressing mode. Five PSW flags are set following this operation, but the operands are not affected.

DST-SRC

DST SRC

Assembly Language Format: CMPL Lreg, Lreg

Object Code Format: [11000101] [src Lreg] [dst#Lreg]

Bytes: 3
States: 7

Flags Affected							
Z	N	V	VT	C	X	I	ST
✓	✓	✓	✓	✓	X	—	—

20. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101] [wreg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

21. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

22. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.
Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

23. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIV	lreg,	waop

Object Code Format: [11111110] [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 29 + CEA (26 + CEA on 80C196KB)

Flags Affected						
Z	N	C	V	VT	ST	
—	—	—	?	↑	—	8096
—	—	—	✓	↑	—	80C196KB

24. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVB	wreg,	baop

Object Code Format: [11111110] [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 21 + CEA (18 + CEA on 80C196KB)

Flags Affected						
Z	N	C	V	VT	ST	
—	—	—	?	↑	—	8096
—	—	—	✓	↑	—	80C196KB

25. DIVU — DIVIDE WORDS

Operation: This instruction divides the content of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order WORD will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVU	lreg,	waop

Object Code Format: [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 25 + CEA (24 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↕	↑	—

26. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination, (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVUB	wreg,	baop

Object Code Format: [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA (16 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↕	↑	—

27. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) <> 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000] [breg] [disp]

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

28. DJNZW — DECREMENT AND JUMP IF NOT ZERO WORD (80C196KB only)[1]

Operation: This instruction is the same as the DJNZ except that the count is a word operand. A counter word is decremented; if the result is not zero the jump is taken. The range of the jump is −128 to +127.

```
COUNT ← COUNT ← −1
if COUNT <> 0 then
PC ← PC + disp (sign extended)
```

Assembly Language Format: DJNZW wreg,cadd

Object Code Format: [11100001] [wreg] [disp]

Bytes: 3
States: jump not taken: 6
jump taken: 10

Flags Affected							
Z	N	V	VT	C	X	I	ST
—	—	—	—	—	X	—	—

NOTE:
1. The DJNZW is not supported on the current stepping of the 80C196KB. The instruction will not generate an unimplemented opcode interrupt.

29. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

```
Interrupt Enable (PSW.9) ← 1
```

Assembly Language Format: EI

Object Code Format: [11111011]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

30. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST) < 8000H then
  (high word DEST) ← 0
else
  (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110] [lreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

31. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST) < 80H then
  (high byte DEST) ← 0
else
  (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

32. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111] [wreg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

33. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

34. IDLPD — IDLE/POWERDOWN (80C196KB only)

Operation: This instruction is used for entry into the idle and powerdown modes. Selecting IDLE or POWERDOWN is done using the key operand. If the operand is not a legal key, the part executes a reset sequence. The bus controller will complete any prefetch cycle in progress before the CPU stops or resets.

```
if KEY = 1 then enter IDLE
else if KEY = 2 then enter
POWERDOWN
else execute reset.
```

Assembly Language Format: IDLPD #key (key is 8-bit value)

Object Code Format: [11110110] [key]

Bytes: 2
States: legal key: 8
illegal key: 25

	Flags Affected							
	Z	N	V	VT	C	x	I	ST
Legal Key:	—	—	—	—	—	X	—	—
Illegal Key:	0	0	0	0	0	X	0	0

(— = Unchanged)

35. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
  PC ← PC + disp (sign-extened to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb] [breg] [disp]
where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

36. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg] [disp]
where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

37. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

38. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

39. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

40. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

41. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

42. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

43. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

44. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

45. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111 [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

46. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If the carry flag is set (i.e., 1) and the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

47. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

48. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

49. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jumps Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

50. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000] [disp]

Bytes: 2
State: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

51. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

52. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow trap flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

53. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP − 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111] [disp-low] [disp-hi]

Bytes: 3
States: Onchip stack: 13 (11 on 80C196KB)
Offchip stack: 16 (13 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

54. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

	DST	SRC
LD	wreg,	waop

Object Code Format: [101000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

55. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

	DST	SRC
LDB	breg,	baop

Object Code Format: [101100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

56. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
    (high byte DEST) ← 0
else
    (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
        DST    SRC
LDBSE   wreg,  baop
```

Object Code Format: [101111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

57. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
        DST    SRC
LDBZE   wreg,  baop
```

Object Code Format: [101011aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

58. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111] [disp-low] [disp-hi]

Bytes: 3
States: 8 (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

59. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MUL	lreg,	waop

Object Code Format: [11111110] [011011aa] [waop] [lreg]

Bytes: 3 + BEA
States 29 + CEA (16 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

60. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MUL	lreg,	wreg,	waop

Object Code Format: [11111110] [010011aa] [waop] [wreg] [lreg]

Bytes: 4 + BEA
States: 30 + CEA (16 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

61. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULB	wreg,	baop

Object Code Format: [11111110] [011111aa] [baop] [wreg]

Bytes: 3 + BEA
States: 21 + CEA (12 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

62. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULB	wreg,	breg	baop

Object Code Format: [11111110] [010111aa] [baop] [breg] [wreg]

Bytes: 4 + BEA
States: 22 + CEA (12 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

63. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULU	lreg,	waop

Object Code Format: [011011aa] [waop] [lreg]

Bytes: 2 + BEA
States: 25 + CEA (14 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

64. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULU	lreg,	wreg,	waop

Object Code Format: [010011aa] [waop] [wreg] [lreg]

Bytes: 3 + BEA
States: 26 + CEA (14 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

65. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULUB	wreg,	baop

Object Code Format: [011111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA (10 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

66. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULUB	wreg,	breg,	baop

Object Code Format: [010111aa] [baop] [breg] [wreg]

Bytes: 3 + BEA
States: 18 + CEA (12 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

67. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011] [wreg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

68. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.
(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

69. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

70. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111] [breg] [lreg]

Bytes: 3
States: 11 + No. of shifts performed

Flags Affected					
Z	N	C	V	VT	ST
✓	?	0	—	—	—

71. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT (DEST)
```

Assembly Language Format: NOT wreg

Object Code Format: [00000010] [wreg]

Bytes 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

72. NOTB — COMPLEMENT BYTE

Operation: The vaule of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

73. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

	DST	SRC
OR	wreg,	waop

Object Code Format: [100000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

74. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

75. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa] [waop]

Bytes 1 + BEA
States: Onchip Stack: 12 + CEA (8 + CEA on 80C196KB)
Offchip Stack: 14 + CEA (11 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

76. POPA — POP ALL (80C196KB only)

Operation: This instruction is used instead of POPF to support the 8 additional interrupts. It is similar to POPF, but pops two words instead of one. The first word is popped into the INT_MASK1/WSR register pair, while the second word is popped into the PSW/INT_MASK register pair. As a result of this instruction the SP is incremented by 4. Interrupts cannot occur between this instruction and the one following it.

```
INT_MASK1/WSR ← (SP)
SP ← SP + 2
PSW/INT_MASK ← (SP)
SP ← SP + 2
```

Assembly Language Format: POPA

Object Code Format: [11110101]

Bytes: 1
States: Onchip Stack: 12
Offchip Stack: 18

Flags Affected							
Z	N	V	VT	C	X	I	ST
✓	✓	✓	✓	✓	X	✓	✓

(✓ = changed)

77. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

```
(PSW) ← (SP)
SP ← SP + 2
```

Assembly Language Format: POPF

Object Code Format: [11110011]

Bytes: 1
States: Onchip Stack: 9 (7 on 80C196KB)
Offchip Stack: 13 (10 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

78. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

```
SP ← SP − 2
(SP) ← (DEST)
```

Assembly Language Format: PUSH waop

Object Code Format: [110010aa] [waop]

Bytes: 1 + BEA
States: Onchip Stack: 8 + CEA (6 + CEA on 80C196KB)
Offchip Stack: 12 + CEA (8 + CEA on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

79. PUSHA — PUSH ALL (80C196KB only)

Operation: This instruction is used instead of PUSHF to support the 8 additional interrupts. It is similar to PUSHF, but pushes two words instead of one. The first word pushed is the same as for the PUSHF instruction, PSW/INT_MASK. The second word pushed is formed by the INT_MASK1/WSR register pair. As a result of this instruction the PSW, INT_MASK, and INT_MASK1 registers are cleared, and the SP is decremented by 4. Interrupts are disabled in two ways by this instruction since both PSW.9 and the interrupt masks are cleared. Interrupts cannot occur between this instruction and the one following it.

```
SP ← SP − 2
(SP) ← PSW/INT_MASK
PSW/INT_MASK ← 0
SP ← SP − 2
(SP) ← INT_MASK1/WSR
INT_MASK1 ← 0
```

Assembly Language Format: PUSHA

Object Code Format: [11110100]

Bytes: 1
States: Onchip Stack: 12
Offchip Stack: 18

Flags Affected							
Z	N	V	VT	C	X	I	ST
0	0	0	0	0	X	0	0

80. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

```
SP ← SP − 2
(SP) ← PSW
PSW ← 0
```

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Bytes: 1
States: Onchip Stack: 8 (6 on 80C196KB)
Offchip Stack: 12 (8 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

81. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

```
PC ← (SP)
SP ← SP + 2
```

Assembly Language Format: RET

Object Code Format: [11110000]

Bytes: 1
States: Onchip Stack: 12 (11 on 80C196KB)
Offchip Stack: 16 (14 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

82. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value. Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Bytes: 1
States: 16 (15 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

83. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP − 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx] [disp-low]

where xxx holds the three high-order bits of displacement.

Bytes: 2
States Onchip Stack: 13 (11 on 80C196KB)
Offchip Stack: 16 (13 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

84. SETC — SET CARRY FLAG

Operation: The carry flag is set.

```
C ← 1
```

Assembly Language Format: SETC

Object Code Format: [11111001]

Bytes: 1
States: 4 (2 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	1	—	—	—

85. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
SHL   wreg,#count
```

or

```
SHL   wreg,breg
```

Object Code Format: [00001001] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

86. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    TEMP ← Temp – 1
end_while
```

Assembly Language Format:

```
        SHLB    breg, #count
or
        SHLB    breg,breg
```

Object Code Format: [00011001] [cnt/breg] [breg]

Bytes 3
States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

87. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
SHLL    lreg, #count
or
SHLL    lreg, breg
```

Object Code Format: [00001101] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

88. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp – 1
end_while
```

Assembly Language Format:

```
        SHR     wreg,#count
or
        SHR     wreg,breg
```

Object Code Format: [00001000] [cnt/breg] [wreg]

Bytes: 3

States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

89. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRA    wreg,#count
or
        SHRA    wreg,breg
```

Object Code Format: [00001010] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

90. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
SHRAB   breg,#count
or
SHRAB   breg,breg
```

Object Code Format: [00011010] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

91. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAL   lreg,#count
or
        SHRAL   lreg,breg
```

Object Code Format: [00001110] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

92. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
SHRB    breg,#count
or
SHRB    breg,breg
```

Object Code Format: [00011000] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed (6 + No. on 80C196KB)
note: 0 place shifts take 8 states. (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

93. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DSET) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
SHRL    lreg,#count
or
SHRL    lreg,breg
```

Object Code Format: [00001100] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

94. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx] [disp-low]
where xxx holds the three high order bits of the displacement.

Bytes: 2
States: 8 (7 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

95. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000] [breg]

Bytes: 2
States: 4 (3 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

96. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
ST	wreg,	waop

Object Code Format: [110000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

97. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
STB	breg,	baop

Object Code Format: [110001aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

98. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUB     wreg,   waop
```

Object Code Format: [011010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

99. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2,
SUB     wreg,   wreg,   waop
```

Object Code Format: [010010aa] [waop] [Sweg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

100. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUBB    breg,   baop
```

Object Code Format: [011110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

101. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the second byte operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
SUBB    breg,   Sbreg   baop
```

Object Code Format: [010110aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

102. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the orignal destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBC	wreg,	waop

Object Code Format: [101010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

103. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBCB	breg,	baop

Object Code Format: [101110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

104. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

```
SP ← SP − 2
(SP) ← PC
PC ← (2010H)
```

Assembly Language Format: This instruction is not supported by revision 1.0 of the 8096 assembly language.

Object Code Format: [11110111]

Bytes:		1
States	Onchip Stack:	21 (16 on 80C196KB)
	Offchip Stack:	24 (18 on 80C196KB)

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

105. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XOR     wreg,   waop
```

Object Code Format: [100001aa] [waop] [wreg]

Bytes:	2 + BEA
States:	4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

106. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XORB    breg,   baop
```

Object Code Format: [100101aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

Using the 8096

5

5 Using the 8096

USING THE 8096

1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS®-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

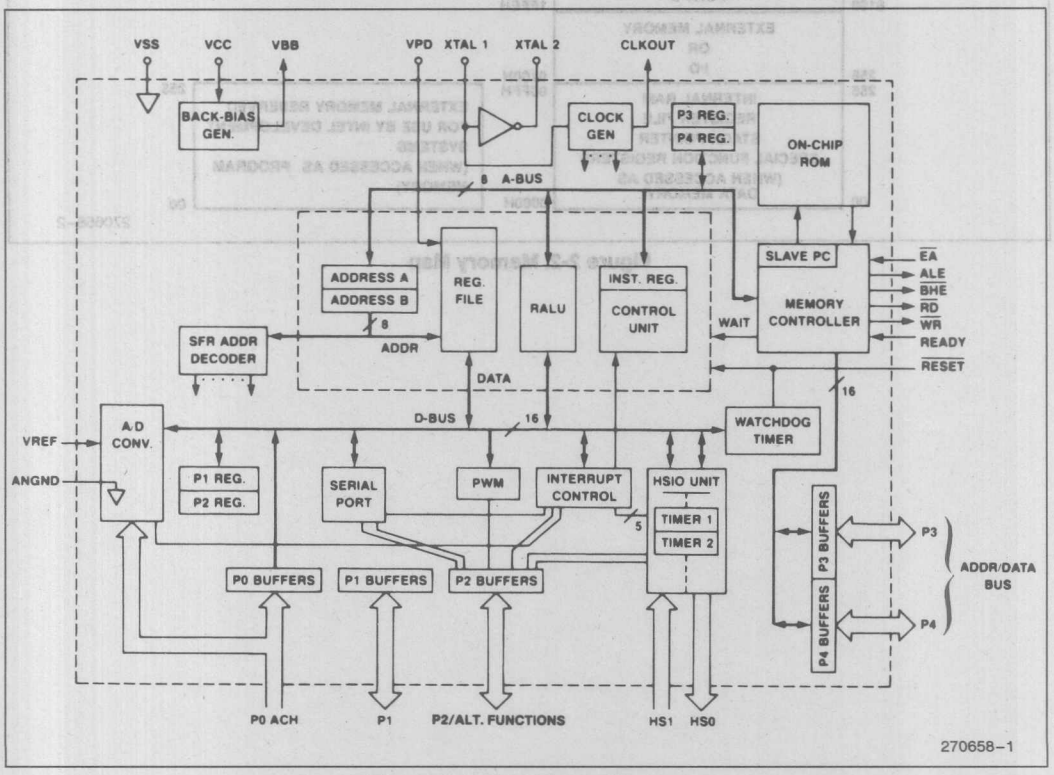


Figure 2-1. 8096 Block Diagram

2.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.

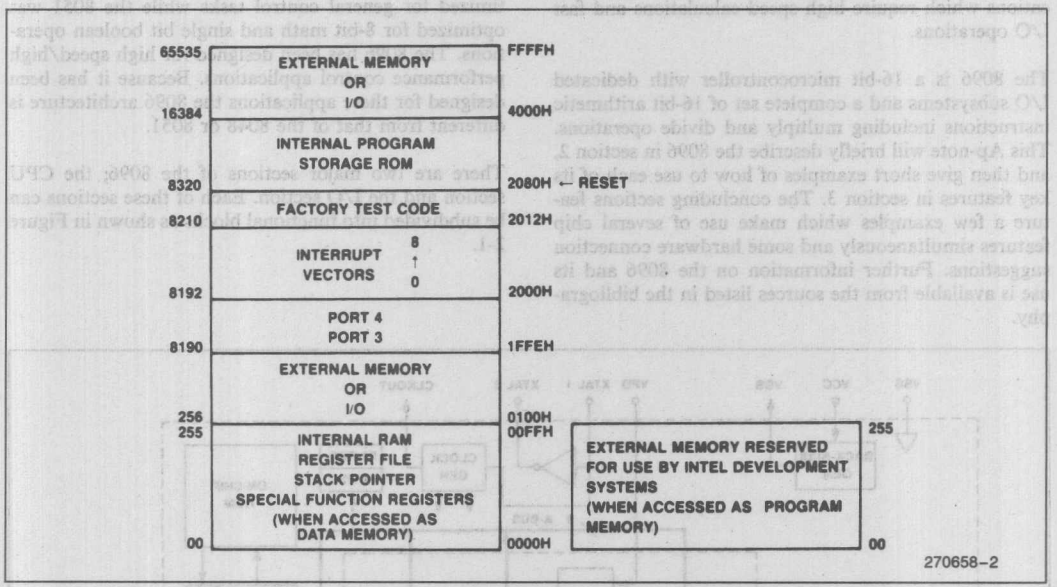


Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written to. More information about the use of these registers is included in the description of the features which they control.

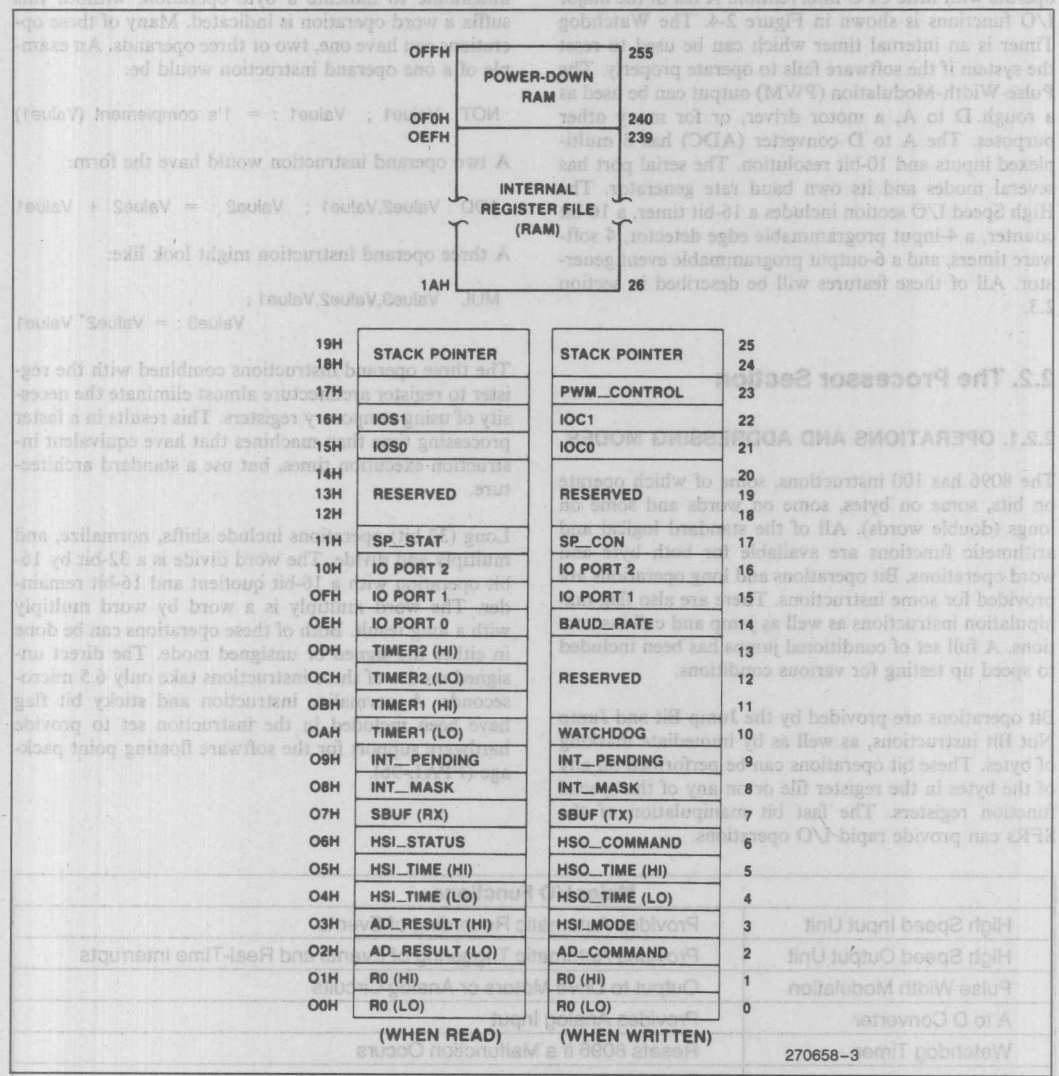


Figure 2-3: SFR Layout

2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

```
NOT   Value1 ;  Value1 : = 1's complement (Value1)
```

A two operand instruction would have the form:

```
ADD   Value2,Value1 ;  Value2 : = Value2 + Value1
```

A three operand instruction might look like:

```
MUL   Value3,Value2,Value1 ;
                        Value3 : = Value2* Value1
```

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfunction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	—	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	—	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↕	↕	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↕	?	↕	↕	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↕	?	↕	0	—	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↕	↕	↕	0	—	↕	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	↕	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

```
Mnem   Dest or Src1                  ; One operand direct
Mnem   Dest, Src1                    ; Two operand direct
Mnem   Dest, Src1, Src2              ; Three operand direct

Mnem   #Src1                         ; One operand immediate
Mnem   Dest, #Src1                   ; Two operand immediate
Mnem   Dest, Src1, #Src2             ; Three operand immediate

Mnem   [addr]                        ; One operand indirect
Mnem   [addr]+                       ; One operand indirect auto-increment
Mnem   Dest, [addr]                  ; Two operand indirect
Mnem   Dest, [addr]+                 ; Two operand indirect auto-increment
Mnem   Dest, Src1, [addr]            ; Three operand indirect
Mnem   Dest, Src1, [addr]+           ; Three operand indirect auto-increment

Mnem   Dest, offs [addr]             ; Two operand indexed (short or long)
Mnem   Dest, Src1, offs [addr]       ; Three operand indexed (short or long)

Where: "Mnem" is the instruction mnemonic
       "Dest" is the destination register
       "Src1", "Src2" are the source registers
       "addr" is a register containing a value to be used in computing the address of an operand
       "offs" is an offset used in computing the address of an operand
```

270658-B3

Figure 2-6. Instruction Format

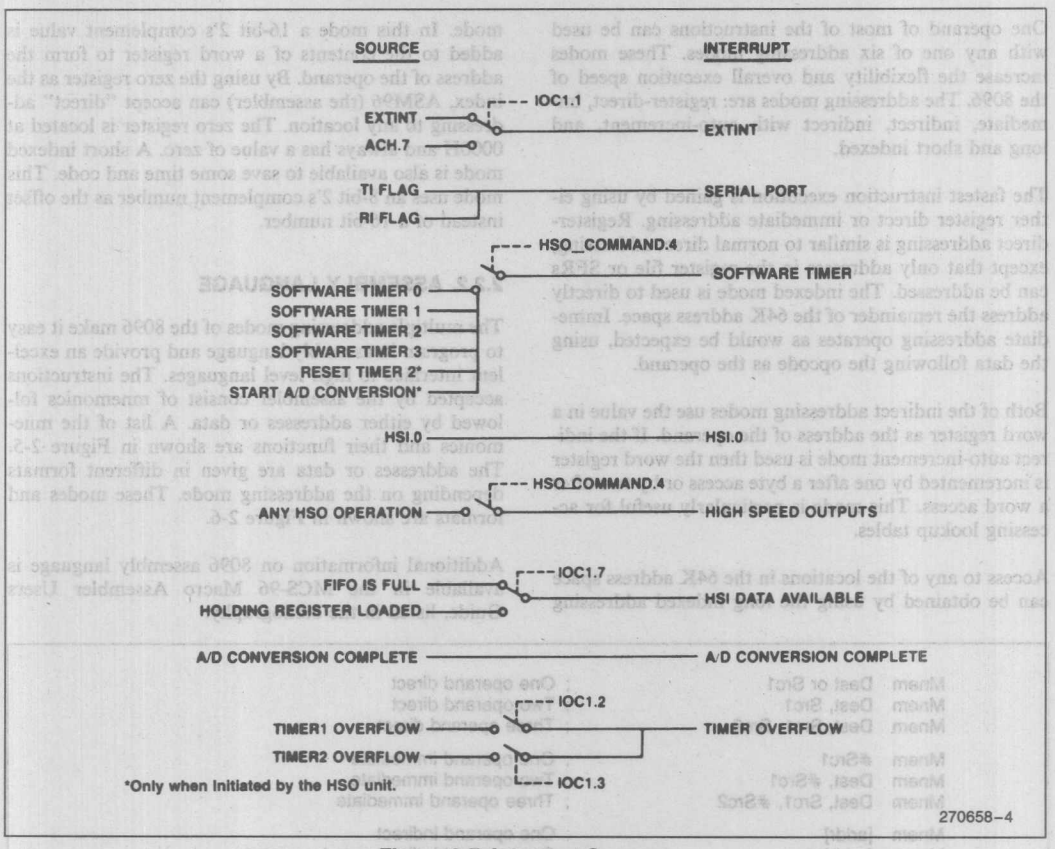


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

source. When the vector is taken the INT_PENDING bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

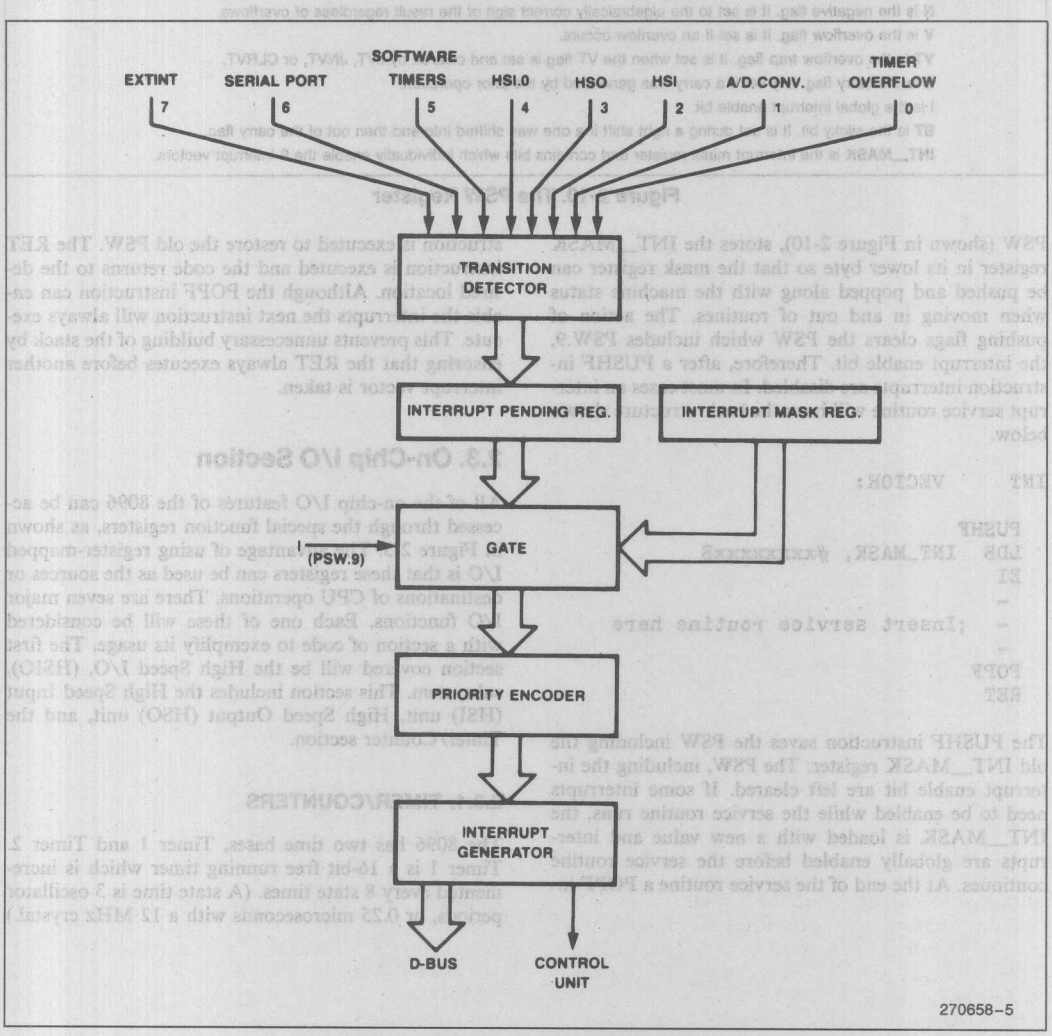


Figure 2-9. Interrupt Structure Block Diagram

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	INT_MASK							

WHERE:

Z is the zero flag. It is set when the result of an operation is zero.

N is the negative flag. It is set to the algebraically correct sign of the result regardless of overflows.

V is the overflow flag. It is set if an overflow occurs.

VT is the overflow trap flag. It is set when the VT flag is set and cleared by JVT, JNVT, or CLRVT.

C is the carry flag. It is set if a carry was generated by the prior operation.

I is the global interrupt enable bit.

ST is the sticky bit. It is set during a right shift if a one was shifted into and then out of the carry flag.

INT_MASK is the interrupt mask register and contains bits which individually enable the 8 interrupt vectors.

Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

```
INT     VECTOR:

  PUSHF
  LDB   INT_MASK, #xxxxxxxxB
  EI
   -
   -   ;Insert service routine here
   -
  POPF
  RET
```

The PUSHF instruction saves the PSW including the old INT_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF instruction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)

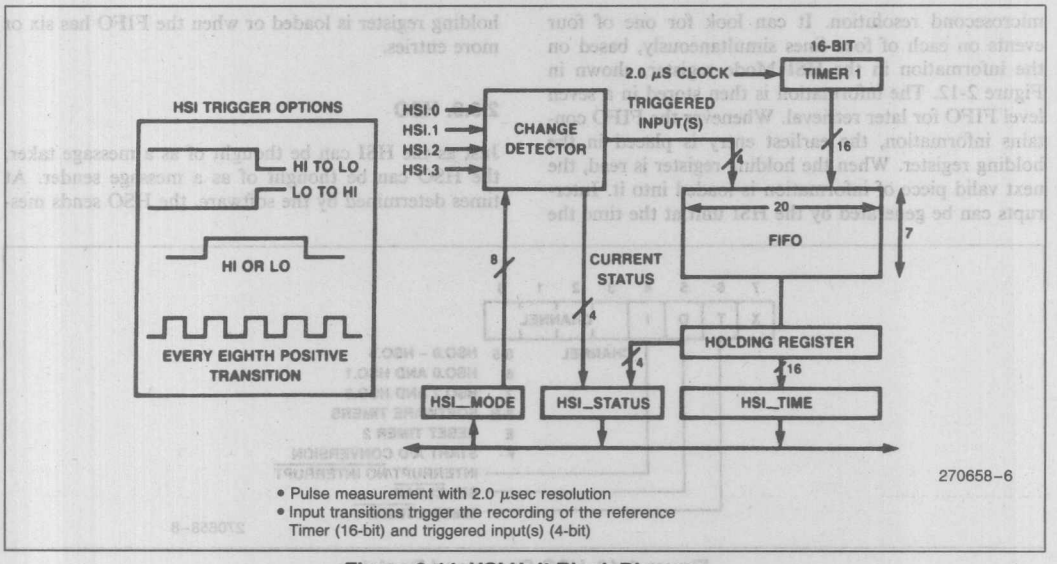


Figure 2-11. HSI Unit Block Diagram

Its value can be read at any time and used as a reference for both the HSI section and the HSO section. Timer 1 can cause an interrupt when it overflows, and cannot be modified or stopped without resetting the entire chip. Timer 2 is really an event counter since it uses an external clock source. Like Timer 1, it is 16-bits wide, can be read at any time, can be used with the HSO section, and can generate an interrupt when it overflows. Control of Timer 2 is limited to incrementing it and resetting it. Specific values can not be written to it.

Although the 8096 has only two timers, the timer flexibility is equal to a unit with many timers thanks to the HSIO unit. The HSI enables one to measure times of external events on up to four lines using Timer 1 as a timer base. The HSO unit can schedule and execute internal events and up to six external events based on the values in either Timer 1 or Timer 2. The 8096 also includes separate, dedicated timers for the baud rate generator and watchdog timer.

2.3.2. HSI

The HSI unit can be thought of as a message taker which records the line which had an event and the time at which the event occurred. Four types of events can trigger the HSI unit, as shown in the HSI block diagram in Figure 2-11. The HSI unit can measure pulse widths and record times of events with a 2

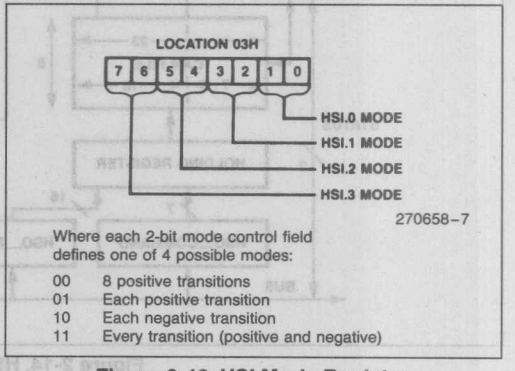


Figure 2-12. HSI Mode Register

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the holding register is loaded or when the FIFO has six or more entries.

2.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

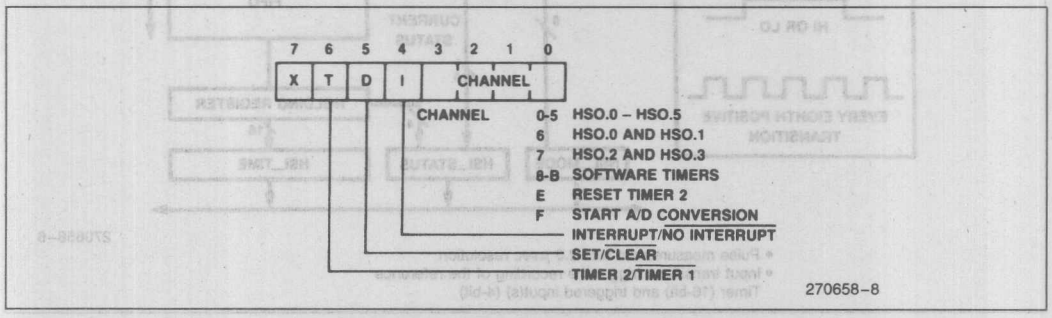


Figure 2-13. HSO Command Register

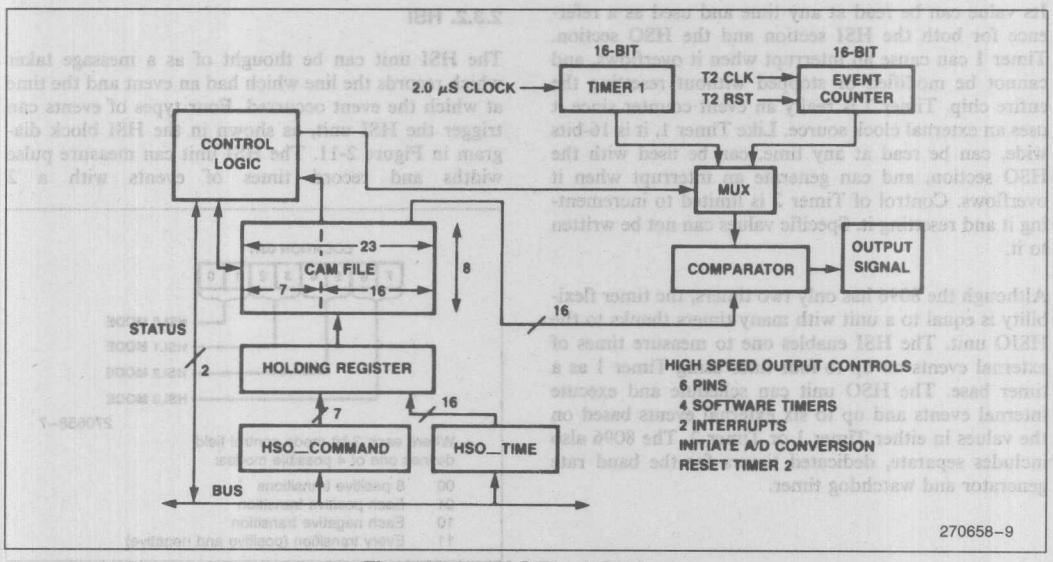


Figure 2-14. HSO Block Diagram

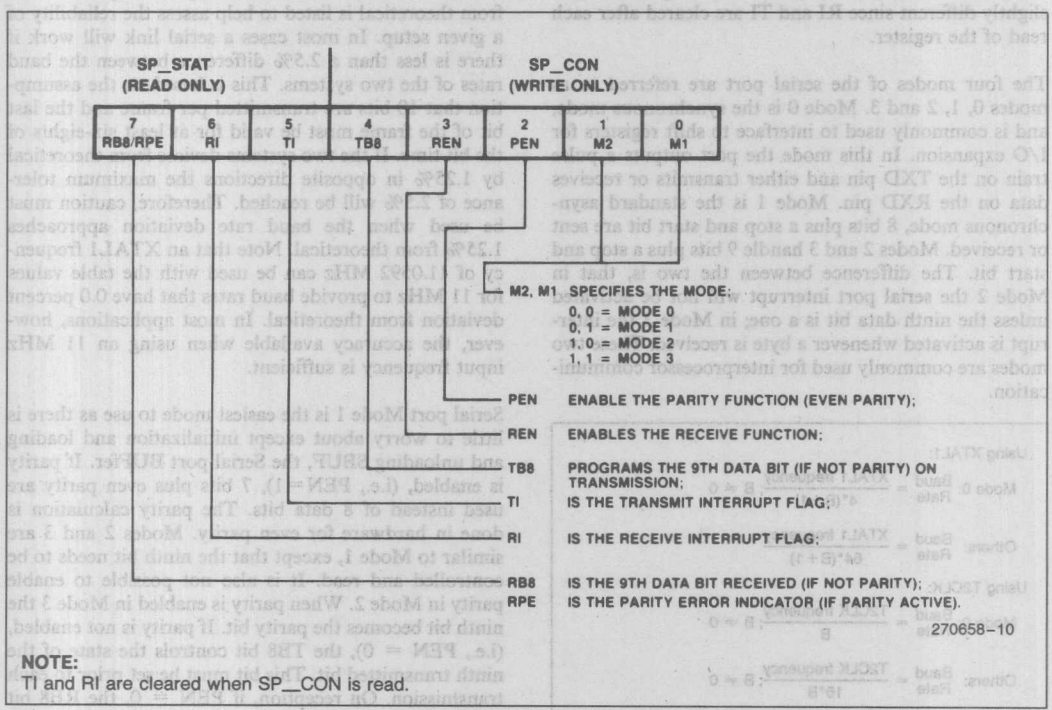
start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.



NOTE:
TI and RI are cleared when SP_CON is read.

Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONtrol/Serial Port STATus) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:

Mode 0: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$

Using T2CLK:

Mode 0: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}; B \neq 0$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eights of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUFfer. If parity is enabled, (i.e., PEN = 1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	**Baud Register Value**	**Percent Error**
19.2K	8009H	+2.40
9600	8013H	+2.40
4800	8026H	−0.16
2400	804DH	−0.16
1200	809BH	−0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+0.54
9600	8011H	+0.54
4800	8023H	+0.54
2400	8047H	+0.54
1200	808EH	−0.16
300	823CH	+0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	−1.70
9600	800FH	−1.70
4800	8020H	+1.38
2400	8040H	−0.16
1200	8081H	−0.16
300	8208H	+0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communicaton can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

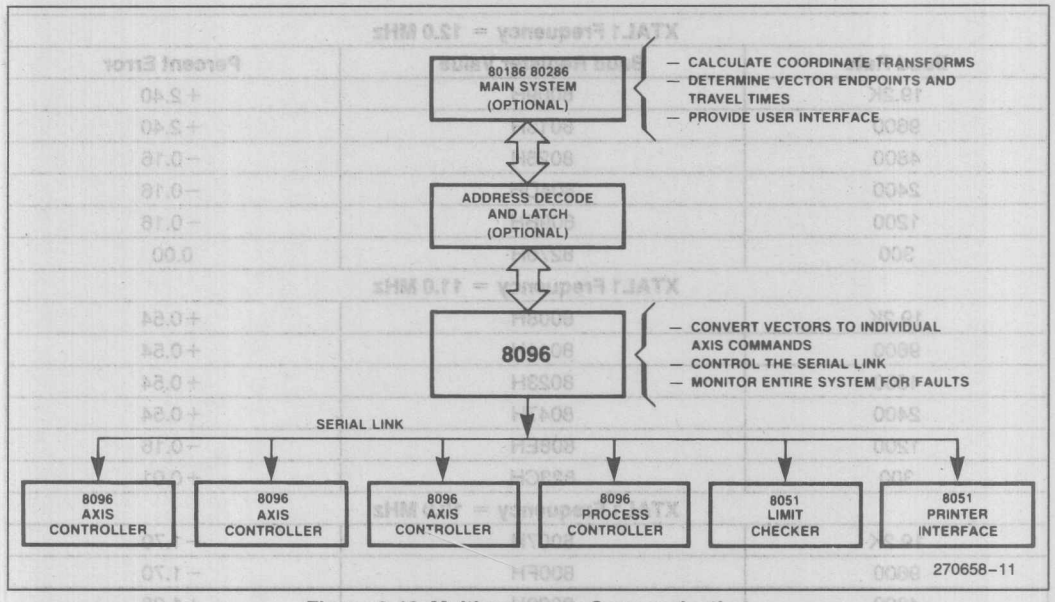


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

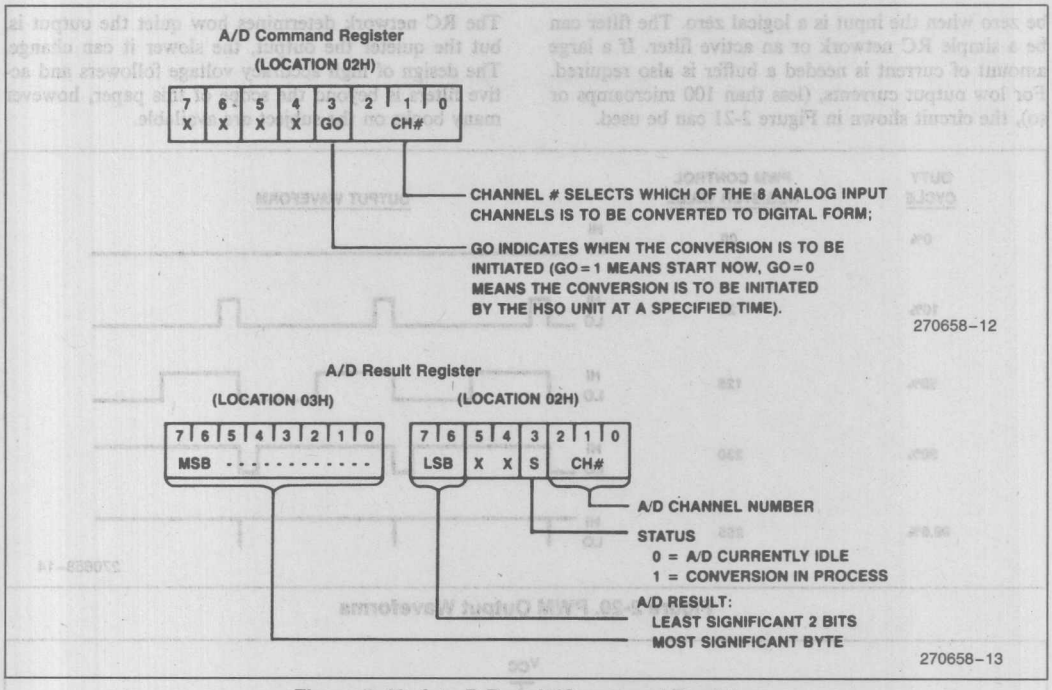


Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD_RESULT/AD_COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will

be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

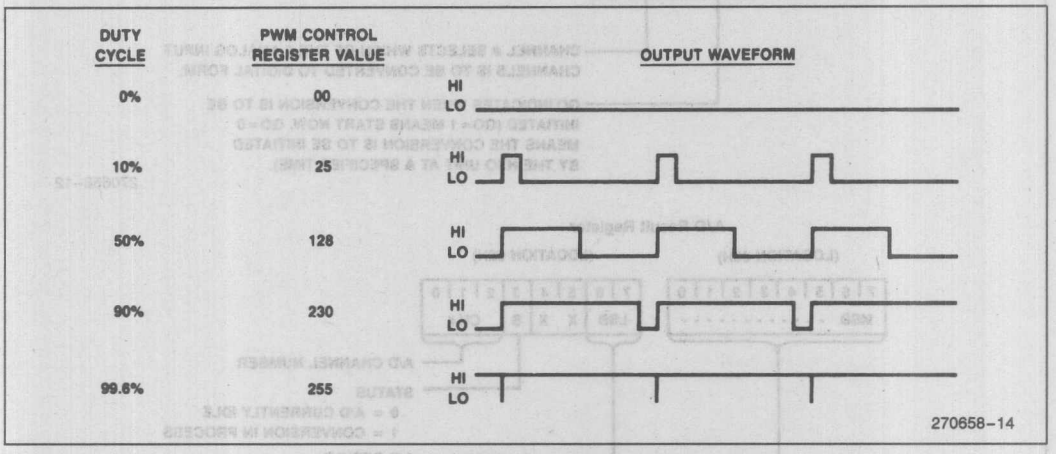


Figure 2-20. PWM Output Waveforms

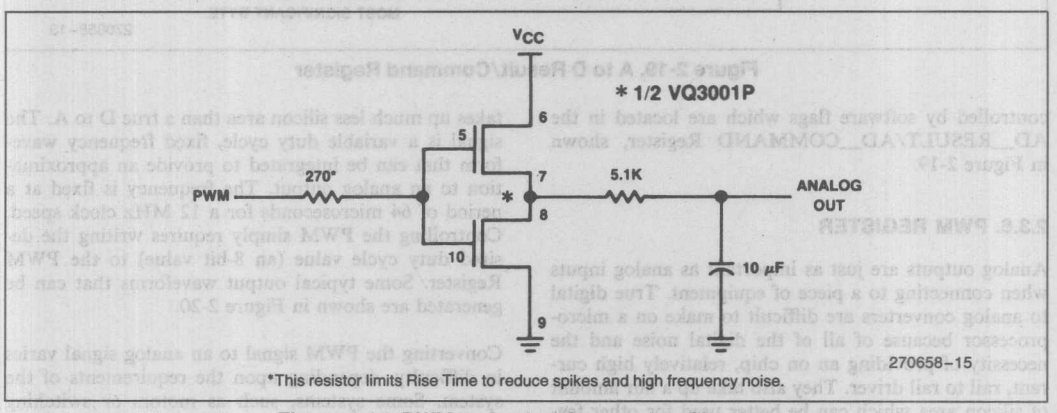


Figure 2-21. PWM to Analog Conversion Circuitry

3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the labels in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

```
;*******************************************************************************
;
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
;
;*******************************************************************************
;
ZERO            EQU   00h:WORD      ; R/W
AD_COMMAND      EQU   02H:BYTE      ;   W
AD_RESULT_LO    EQU   02H:BYTE      ; R
AD_RESULT_HI    EQU   03H:BYTE      ; R
HSI_MODE        EQU   03H:BYTE      ;   W
HSO_TIME        EQU   04H:WORD      ;   W
HSI_TIME        EQU   04H:WORD      ; R
HSO_COMMAND     EQU   06H:BYTE      ;   W
HSI_STATUS      EQU   06H:BYTE      ; R
SBUF            EQU   07H:BYTE      ; R/W
INT_MASK        EQU   08H:BYTE      ; R/W
INT_PENDING     EQU   09H:BYTE      ; R/W
SPCON           EQU   11H:BYTE
SPSTAT          EQU   11H:BYTE
WATCHDOG        EQU   0AH:BYTE      ;   W  WATCHDOG TIMER
TIMER1          EQU   0AH:WORD      ; R
TIMER2          EQU   0CH:WORD      ; R
PORT0           EQU   0EH:BYTE      ; R
BAUD_REG        EQU   0EH:BYTE      ;   W
PORT1           EQU   0FH:BYTE      ; R/W
PORT2           EQU   10H:BYTE      ; R/W
IOC0            EQU   15H:BYTE      ;   W
IOS0            EQU   15H:BYTE      ; R
IOC1            EQU   16H:BYTE      ;   W
IOS1            EQU   16H:BYTE      ; R
PWM_CONTROL     EQU   17H:BYTE      ;   W
SP              EQU   18H:WORD      ; R/W STACK POINTER


RSEG at 1CH

        AX:     DSW     1
        DX:     DSW     1
        BX:     DSW     1
        CX:     DSW     1

        AL      EQU     AX      :BYTE
        AH      EQU     (AX+1)  :BYTE
```

270658-16

Listing 3-1. Include File DEMO.96.INC

In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{UpperOutput-Lower Output}}{\text{Upper Input-Lower Input}} * (\text{Actual Input-Lower Input})$$

$$\text{Actual Output} = \text{Lower Output} + \text{Delta Out}$$

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900\text{-}400}{300\text{-}200} * (259\text{-}200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

$$\text{Actual Output} = 400 + 295 = 695$$

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 (2**4) was selected. To cover the input range 17 words are needed, 255/16 + 1 word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600

```
$TITLE('INTER1.APT: Interpolation routine 1')
;;;;;;;   8096 Assembly code for table lookup and interpolation

$INCLUDE(:F1:DEMO96.INC)        ; Include demo definitions


RSEG  at 22H

        IN_VAL:         dsb     1               ; Actual Input Value
        TABLE_LOW:      dsw     1
        TABLE_HIGH:     dsw     1
        IN_DIF:         dsw     1               ; Upper Input - Lower Input
        IN_DIFB         equ     IN_DIF  :byte
        TAB_DIF:        dsw     1               ; Upper Output - Lower Output
        OUT:            dsw     1
        RESULT:         dsw     1
        OUT_DIF:        dsl     1               ; Delta Out


CSEG at 2080H

        LD      SP, #100H
```

270658-17

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

```
look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
        SHRB    AL, #3          ; Divide the byte by 8
        ANDB    AL, #11111110B  ; Insure AL is a word address
                                ;  This effectively divides AL by 2
                                ;  so AL = IN_VAL/16

        LDBZE   AX, AL          ; Load byte AL to word AX
        LD      TABLE_LOW, TABLE [AX]    ; TABLE_LOW is loaded with the value
                                         ;  in the table at table location AX

        LD      TABLE_HIGH, (TABLE+2)[AX]  ; TABLE_HIGH is loaded with the
                                           ; value in the table at table
                                           ; location AX+2
                                           ; (The next value in the table)

        SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                                           ; TAB_DIF=TABLE_HIGH-TABLE_LOW

        ANDB    IN_DIFB, IN_VAL, #0FH      ; IN_DIFB=least significant 4 bits
                                           ;  of IN_VAL
        LDBZE   IN_DIF, IN_DIFB            ; Load byte IN_DIFB to word IN_DIF

        MUL     OUT_DIF, IN_DIF, TAB_DIF
                                           ; Output_difference =
                                           ;  Input_difference*Table_difference
        SHRAL   OUT_DIF, #4                ; Divide by 16 (2**4)

        ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                        ;  generated with truncated IN_VAL
                                        ;  as input
        SHRA    OUT, #4                 ; Round to 12-bit answer

        ADDC    OUT, zero               ; Round up if Carry = 1

no_inc: ST      OUT, RESULT             ; Store OUT to RESULT

        BR      look                    ; Branch to "look:"


cseg    AT 2100H

table:  DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
        DCW     5D00H, 6A00H, 7200H, 7800H
        DCW     7B00H, 7D00H, 7600H, 6D00H
        DCW     5D00H, 4B00H, 3400H, 2200H
        DCW     1000H

END
```

270658-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

```
$TITLE('INTER2.APT: Interpolation routine 2')

;;;;;;;   8096 Assembly code for table lookup and interpolation
;;;;;;;   Using tabled values in place of division

$INCLUDE(:F1:DEMO96.INC) ; Include demo definitions

RSEG  at 24H

        IN_VAL:         dsb     1           ; Actual Input Value
        TABLE_LOW:      dsw     1           ; Table value for function
        TABLE_INC:      dsw     1           ; Incremental change in function
        IN_DIF:         dsw     1           ; Upper Input - Lower Input
        IN_DIFB         equ     IN_DIF  :byte
        OUT:            dsw     1
        RESULT:         dsw     1
        OUT_DIF:        dsl     1           ; Delta Out
```

270658-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```
CSEG at 2080H

        LD      SP, #100H       ; Initialize SP to top of reg. file

look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
        SHRB    AL, #3          ; Divide the byte by 8
        ANDB    AL, #11111110B  ; Ensure AL is a word address
                                ;  This effectively divides AL by 2
                                ;  so AL = IN_VAL/16
        LDBZE   AX, AL          ; Load byte AL to word AX

        LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                         ;  in the value table at location AX

        LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                         ; in the increment table at
                                         ; location AX

        ANDB    IN_DIFB, IN_VAL, #0FH    ; IN_DIFB=least significant 4 bits
                                         ;  of IN_VAL
        LDBZE   IN_DIF, IN_DIFB          ; Load byte IN_DIFB to word IN_DIF

        MUL     OUT_DIF, IN_DIF, TABLE_INC
                                         ; Output_difference =
                                         ;  Input_difference*Incremental_change

        ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                         ;  generated with truncated IN_VAL
                                         ;  as input
        SHR     OUT, #4                  ; Round to 12-bit answer
        ADDC    OUT, zero                ; Round up if Carry = 1

no_inc: ST      OUT, RESULT              ; Store OUT to RESULT
        BR      look                     ; Branch to "look:"

cseg    AT 2100H

val_table:
        DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
        DCW     5D00H, 6A00H, 7200H, 7800H
        DCW     7B00H, 7D00H, 7600H, 6D00H
        DCW     5D00H, 4B00H, 3400H, 2200H
        DCW     1000H
inc_table:
        DCW      0200H,  0140H,  0180H,  0110H   ; Table of incremental
        DCW      00D0H,  0080H,  0060H,  0030H   ; differences
        DCW     00020H, 0FF90H, 0FF70H, 0FF00H
        DCW     0FEE0H, 0FE90H, 0FEE0H, 0FEE0H

END
```

270658-20

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocator and linker. The command line to be used is:

```
RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)
```

```
/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

PLMEX:    DO;

DECLARE IN_VAL          WORD        PUBLIC;
DECLARE TABLE_LOW       INTEGER     PUBLIC;
DECLARE TABLE_HIGH      INTEGER     PUBLIC;
DECLARE TABLE_DIF       INTEGER     PUBLIC;
DECLARE OUT             INTEGER     PUBLIC;
DECLARE RESULT          INTEGER     PUBLIC;
DECLARE OUT_DIF         LONGINT     PUBLIC;
DECLARE TEMP            WORD        PUBLIC;

DECLARE TABLE(17)       INTEGER  DATA (
        0000H, 2000H, 3400H, 4C00H,       /* A random function */
        5D00H, 6A00H, 7200H, 7800H,
        7B00H, 7D00H, 7600H, 6D00H,
        5D00H, 4B00H, 3400H, 2200H,
        1000H);

DMPY:   PROCEDURE (A,B) LONGINT EXTERNAL;
        DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
    TEMP=SHR(IN_VAL,4);      /* TEMP is the most significant 4 bits of IN_VAL */

    TABLE_LOW=TABLE(TEMP);      /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
    TABLE_HIGH=TABLE(TEMP+1);   /* The code would work but the 8096 would     */
                                /* do two shifts                              */

    TABLE_DIF=TABLE_HIGH-TABLE_LOW;

    OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

    OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                       in this case 4 places are shifted      */

    IF CARRY=0 THEN RESULT=OUT;  /* Using the hardware flags must be done   */
      ELSE  RESULT=OUT+1;        /* with care to ensure the flag is tested  */
                                 /* in the desired instruction sequence     */
GOTO LOOP;

/* END OF PLM-96 CODE  */

END;
```

270658-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1

```
    $TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')

            SP      EQU     18H:word

    rseg
            EXTRN   PLMREG  :long

    cseg
            PUBLIC  DMPY            ; Multiply two integers and return a
                                    ; longint result in AX, DX registers

    DMPY:   POP     PLMREG+4                ; Load return address
            POP     PLMREG                  ; Load one operand
            MUL     PLMREG,[SP]+    ; Load second operand and increment SP

            BR      [PLMREG+4]          ; Return to PLM code.
    END
```

270658-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```
$TITLE('PULSE.APT: Measuring pulses using the HSI unit')

$INCLUDE(DEMO96.INC)
rseg    at  28H

        HIGH_TIME:      dsw     1
        LOW_TIME:       dsw     1
        PERIOD:         dsw     1
        HI_EDGE:        dsw     1
        LO_EDGE:        dsw     1


cseg    at  2080H

        LD      SP, #100H
        LDB     IOC0, #00000001B        ; Enable HSI 0
        LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge

wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME
        JBS     IOS1, 6, contin         ; If FIFO is full
        JBC     IOS1, 7, wait   ; Wait while no pulse is entered

contin: LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                        ;   HSI_TIME clears HSI_STATUS

        LD      BX, HSI_TIME            ; Load the HSI_TIME

        JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high

hsi_lo: ST      BX, LO_EDGE
        SUB     HIGH_TIME, LO_EDGE, HI_EDGE
        BR      wait


hsi_hi: ST      BX, HI_EDGE
        SUB     LOW_TIME, HI_EDGE, LO_EDGE
        BR      wait

        END
```

270658-23

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register dec-

```
$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')

$INCLUDE(DEMO96.INC)

RSEG AT 28H

        TIME:           DSW 1
        LAST_RISE:      DSW 1
        LAST_FALL:      DSW 1
        HSI_S0:         DSB 1
        IOS1_BAK:       DSB 1
        PERIOD:         DSW 1
        LOW_TIME:       DSW 1
        HIGH_TIME:      DSW 1
        COUNT:          DSW 1

cseg    at      2080H

init:   LD      SP,#100H
        LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                ; Enable PWM,TXD,TIMER1_OVRFLOW_INT

        LDB     HSI_MODE,#10011001B     ; set hsi.1 -; hsi.0 +
        LDB     IOC0,#00000111B         ; Enable  hsi 0,1
                                        ; T2 CLOCK=T2CLK, T2RST=T2RST
                                        ; Clear timer2

wait:   ANDB    IOS1_BAK,#01111111B     ; Clear IOS1_BAK.7
        ORB     IOS1_BAK,IOS1           ; Store into temp to avoid clearing
                                        ; other flags which may be needed
        JBC     IOS1_BAK,7,wait         ; If hsi is not triggered then
                                        ; jump to wait

        ANDB    HSI_S0,HSI_STATUS,#01010101B
        LD      TIME, HSI_TIME

        JBS     HSI_S0,0,a_rise
        JBS     HSI_S0,2,a_fall
        BR      no_cnt

a_rise: SUB     LOW_TIME, TIME, LAST_FALL
        SUB     PERIOD, TIME,LAST_RISE
        LD      LAST_RISE, TIME
        BR      increment

a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
        SUB     PERIOD, TIME,LAST_FALL
        LD      LAST_FALL, TIME

increment:
        INC     COUNT

no_cnt: BR      wait

        END
```

270658-24

Listing 3-7. Enhanced HSI Pulse Measurement Routine

```
$TITLE ('HSOPWM.APT: 8096 EXAMPLE PROGRAM FOR PWM OUTPUTS')

; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
;               HSO_ON_N    HSO on time for pin N
;               HSO_OFF_N   HSO off time for pin N
;
;     Where:    Times are in timer1 cycles
;               N takes values from 0 to 3
;!!!
```

laration section (i.e., the section between "RSEG" and "CSEG") in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a frequency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

```
;     NOTE: Use this file to replace the declaration section of
;           the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
;           the line prior to the label "wait".  Also change the last
;           branch in the program to a "RET".

RSEG

          D_STAT:          DSB       1
          extrn    HSO_ON_0 :word , HSO_OFF_0 :word
          extrn    HSO_ON_1 :word , HSO_OFF_1 :word
          extrn    HSO_TIME :word , HSO_COMMAND :byte
          extrn    TIMER1   :word , IOS0        :byte
          extrn    SP       :word

          public   OLD_STAT
          OLD_STAT:         dsb       1
          NEW_STAT:         dsb       1

cseg
          PUBLIC   wait
```

270658-26

Listing 3-9. Changes to Declarations for HSO Routine

```
          $TITLE('HSODRV.APT: Driver module for HSO PWM program')

HSODRV            MODULE  MAIN, STACKSIZE(8)

                  PUBLIC  HSO_ON_0 , HSO_OFF_0
                  PUBLIC  HSO_ON_1 , HSO_OFF_1
                  PUBLIC  HSO_TIME , HSO_COMMAND
                  PUBLIC  SP, TIMER1 , IOS0

          $INCLUDE(DEMO96.INC)

          rseg at 28H

                  EXTRN   OLD_STAT          :byte

                  HSO_ON_0:         dsw     1
                  HSO_OFF_0:        dsw     1
                  HSO_ON_1:         dsw     1
                  HSO_OFF_1:        dsw     1
                  count:            dsb     1

          cseg  at 2080H

                  EXTRN   wait      :entry

          strt:   DI
                  LD      SP, #100H
                  ANDB    OLD_STAT, IOS0, #0FH
                  XORB    OLD_STAT, #0FH

          initial:
                  LD      CX, #0100H

          loop:   LD      AX, #1000H
                  SUB     BX, AX, CX
                  LD      AX, CX

                  ST      AX, HSO_ON_0
                  ST      BX, HSO_OFF_0
```

270658-27

Listing 3-10. Driver Module for HSO PWM Program

```
        SHR     AX,#1
        SHR     BX,#1
        ST      AX, HSO_ON_1
        ST      BX, HSO_OFF_1

        CALL    wait

        INC     CX
        CMP     CX, #00F00H
        BNE     loop

        BR      initial

        END
```

270658-28

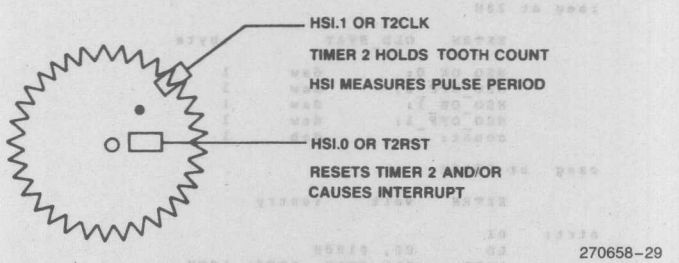
Listing 3-10. Driver Module for HSO PWM Program (Continued)

Since the 8096 needs to keep track of events which often repeat at set intervals it is convenient to be able to have Timer 2 act as a programmable modulo counter. There are several ways of doing this. The first is to program the HSO to reset Timer 2 when Timer 2 equals a set value. A software timer set to interrupt at Timer 2 equals zero could be used to reload the CAM. This software method takes up two locations in the CAM and does not synchronize Timer 2 to the external world.

To synchronize Timer 2 externally the T2 RST (Timer 2 ReSeT) pin can be used. In this way Timer 2 will get reset on each rising edge of T2 RST. If it is desired to have an interrupt generated and time recorded when Timer 2 gets reset, the signal for its reset can be taken from HSI.0 instead of T2RST. The HSI.0 pin has its own interrupt vector which functions independently of the HSI unit.

Another option available is to use the HSI.1 pin to clock Timer 2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer 2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSIO unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

In this system a sequence of events is required based on the position of the gear which represents any piece of rotating machinery. Timer 2 holds the count of the number of tooth edges passed since the index mark. By using HSI.1 as the input to Timer 2, instead of T2 CLK, it is possible to determine tooth count and time information through the HSI. From this information instantaneous velocity and acceleration can be calculated. Having the tooth edge count in Timer 2 means



270658-29

Figure 3-1. Using the HSIO to Monitor Rotating Machinery

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Interprocessor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ±12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.

```
$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')

$INCLUDE(DEMO96.INC)

rseg    at 28H

        CHR:    dsb     1
        SPTEMP: dsb     1
        TEMP0:  dsb     1
        TEMP1:  dsb     1
        RCV_FLAG:       dsb     1

cseg    at 200CH

        DCW     ser_port_int

cseg    at 2080H

        LD      SP, #100H

        LDB     IOC1, #00100000B                ; Set P2.0 to TXD

                ; Baud rate = input frequency / (64*baud_val)
                ; baud_val  = (input frequency/64) / baud rate


baud_val        equ     39              ; 39 = (12,000,000/64)/4800 baud

BAUD_HIGH       equ     ((baud_val-1)/256) OR 80H       ; Set MSB to 1
BAUD_LOW        equ     (baud_val-1) MOD 256


        LDB     BAUD_REG, #BAUD_LOW
        LDB     BAUD_REG, #BAUD_HIGH

        LDB     SPCON, #01001001B       ; Enable receiver, Mode 1

                ; The serial port is now initialized


        STB     SBUF, CHR               ; Clear serial Port
        LDB     TEMP0, #00100000B       ; Set TI-temp

        LDB     INT_MASK, #01000000B    ; Enable Serial Port Interrupt
        EI
loop:   BR      loop            ; Wait for serial port interrupt


ser_port_int:
        PUSHF
rd_again:                               ; This section of code can be replaced
        LDB     SPTEMP, SPSTAT          ; with "ORB  TEMP0, SP_STAT" when the
        ORB     TEMP0, SPTEMP           ; serial port TI and RI bugs are fixed
        ANDB    SPTEMP,#01100000B
        JNE     rd_again        ; Repeat until TI and RI are properly cleared
```

270658-30

Listing 3-11. Using the Serial Port in Mode 1

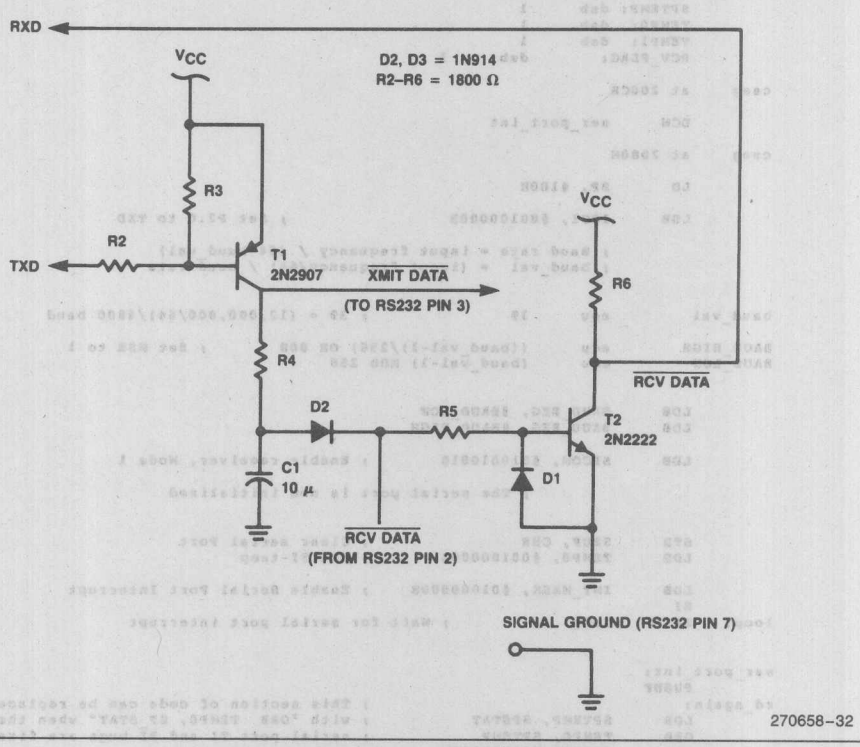
```
get_byte:
        JBC     TEMP0, 6, put_byte      ; If RI-temp is not set
        STB     SBUF, CHR               ; Store byte
        ANDB    TEMP0, #10111111B       ; CLR RI-temp
        LDB     RCV_FLAG, #0FFH         ; Set bit-received flag
put_byte:
        JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
        JBC     TEMP0, 5, continue      ; If TI was not set
        LDB     SBUF, CHR               ; Send byte
        ANDB    TEMP0, #11011111B       ; CLR TI-temp

        ANDB    CHR, #01111111B         ; This section of code appends
        CMPB    CHR,#0DH                ; an LF after a CR is sent
        JNE     clr_rcv
        LDB     CHR, #0AH
        BR      continue
clr_rcv:
        CLRB    RCV_FLAG                ; Clear bit-received flag
continue:
        POPF
        RET

        END
```

270658-31

Listing 3-11. Using the Serial Port in Mode 1 (Continued)



270658-32

Figure 3-2. Serial Port Level Conversion

3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100Ω resistor in series with a 0.22 uf capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which

```
$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')

$INCLUDE(DEMO96.INC)

RSEG    at  28H

        BL      EQU     BX:BYTE
        DL      EQU     DX:BYTE

RESULT_TABLE:
        RESULT_1:       dsw     1
        RESULT_2:       dsw     1
        RESULT_3:       dsw     1
        RESULT_4:       dsw     1


cseg    at  2080H


start:  LD      SP, #100H       ; Set Stack Pointer
        CLR     BX

next:   ADDB    AD_COMMAND,BL, #1000B       ; Start conversion on channel
                                            ; indicated by BL register

        NOP             ; Wait for conversion to start
        NOP
check:  JBS     AD_RESULT_LO, 3, check  ; Wait while A to D is busy

        LDB     AL, AD_RESULT_LO        ; Load low order result
        LDB     AH, AD_RESULT_HI        ; Load high order result

        ADDB    DL, BL, BL              ; DL=BL*2
        LDBZE   DX, DL
        ST      AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2

        INCB    BL              ; Increment BL modulo 4
        ANDB    BL, #03H

        BR      next

        END
```

270658-33

Listing 3-12. Scanning the A to D Channels

order, Port 1 output pins are used to indicate the current status of each task. The actual code listing is included in Appendix B.

The initialization section, shown in Listing 4-1a, clears a few variables and then loads the first set of on and off times to the HSO unit. Note that 8 state times must be waited between consecutive loads of the HSO. If this is not done it is possible to overwrite the contents of the CAM holding register. An A/D interrupt is forced by setting the bit in the Interrupt Pending register. This causes the first A/D interrupt to occur just after the Interrupt Mask register is set and interrupts are enabled.

Listing 4-1. Using Multiple I/O Devices

```
$TITLE ('8096 EXAMPLE PROGRAM FOR PWM OUTPUTS FROM A TO D INPUTS')
$PAGEWIDTH(130)

; This program will provide 3 PWM outputs on HSO pins 0-2
; and one on the PWM.
;
; The PWM values are determined by the input to the A/D converter.
;
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

$INCLUDE(DEMO96.INC)

RSEG AT 28H
        DL      EQU     DX:BYTE

ON_TIME:
        PWM_TIME_1:     DSW     1
        HSO_ON_0:       DSW     1
        HSO_ON_1:       DSW     1
        HSO_ON_2:       DSW     1

RESULT_TABLE:
        RESULT_0:       DSW     1
        RESULT_1:       DSW     1
        RESULT_2:       DSW     1
        RESULT_3:       DSW     1

        NXT_ON_T:       DSW     1
        NXT_OFF_0:      DSW     1
        NXT_OFF_1:      DSW     1
        NXT_OFF_2:      DSW     1
        COUNT:          DSL     1
        AD_NUM:         DSW     1       ; Channel being converted
        TMP:            DSW     1
        HSO_PER:        DSW     1
        LAST_LOAD:      DSB     1

cseg    AT 2000H

        DCW     start           ; Timer_ovf_int
        DCW     Atod_done_int
        DCW     start           ; HSI_data_int
        DCW     HSO_exec_int


cseg    AT 2080H
start:  LD      SP, #100H       ; Set Stack Pointer
        CLR     AX
wait:   DEC     AX              ; wait approx. 0.2 seconds for
        JNE     wait            ; SBE to finish communications

        CLRB    AD_NUM

        LD      PWM_TIME_1, #080H
        LD      HSO_PER, #100H
        LD      HSO_ON_0, #040H
        LD      HSO_ON_1, #080H
        LD      HSO_ON_2, #0C0H

        ADD     NXT_ON_T,  Timer1, #100H
```

270658-34

Listing 4-1a. Initializing the A to D to PWM Program

```
        LDB     HSO_COMMAND, #00110110B       ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T            ; with interrupt
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B       ; Set HSO for timer1, set pin 2
        ADD     HSO_TIME, NXT_ON_T            ; without interrupt

        ORB     LAST_LOAD, #00000111B    ; Last loaded value was set all pins
        LDB     INT_MASK, #00001010B     ; Enable HSO and A/D interrupts
        LDB     INT_PENDING, #00001010B  ; Fake an A/D and HSO interrupt
        EI

loop:   ORB     Port1, #00000001B        ; set P1.0
        ADD     COUNT, #01
        ADDC    COUNT+2,zero
        ANDB    Port1, #11111110B        ; clear P1.0
        BR      loop
```

270658-35

Listing 4-1a. Initializing the A to D to PWM program (Continued)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;    HSO  EXECUTED  INTERRUPT    ;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HSO_exec_int:
        PUSHF
        ORB     Port1, #00000010B        ; Set p1.1

        SUB     TMP,TIMER1, NXT_ON_T
        CMP     TMP,ZERO
        JLT     set_off_times

set_on_times:
        ADD     NXT_ON_T, HSO_PER
        LDB     HSO_COMMAND, #00110110B   ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B   ; Set HSO for timer1, set pin 2
        LD      HSO_TIME, NXT_ON_T

        ORB     LAST_LOAD, #00000111B     ; Last loaded value was all ones

        LDB     PWM_CONTROL, PWM_TIME_1    ; Now is as good a time as any
                                           ; to update the PWM reg
        BR      check_done

set_off_times:
        JBC     LAST_LOAD, 0, check_done

        ADD     NXT_OFF_0 NXT_ON_T, HSO_ON_0
        LDB     HSO_COMMAND, #00010000B      ; Set HSO for timer1, clear pin 0
        LD      HSO_TIME, NXT_OFF_0

        NOP
        ADD     NXT_OFF_1, NXT_ON_T, HSO_ON_1
        LDB     HSO_COMMAND, #00010001B      ; Set HSO for timer1, clear pin 1
        LD      HSO_TIME, NXT_OFF_1

        NOP
        ADD     NXT_OFF_2, NXT_ON_T, HSO_ON_2
        LDB     HSO_COMMAND, #00010010B      ; Set HSO for timer1, clear pin 2
        LD      HSO_TIME, NXT_OFF_2

        ANDB    LAST_LOAD, #11111000B     ; Last loaded value was all 0s

check_done:
        ANDB    Port1, #11111101B         ; Clear P1.1
        POPF
        RET
```

270658-36

Listing 4-1b. Interrupt Driven HSO Routine

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;        A TO D  COMPLETE  INTERRUPT  ;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ATOD_done_int:
        PUSHF
        ORB     Port1, #00000100B            ; Set P1.2

        ANDB    AL, AD_RESULT_LO,#11000000B     ; Load low order result
        LDB     AH, AD_RESULT_HI                ; Load high order result
        ADDB    DL, AD_NUM, AD_NUM              ; DL= AD_NUM *2
        LDBZE   DX, DL
        ST      AX, RESULT_TABLE[DX]         ; Store result indexed by DX

        CMPB    AL, #01000000B
        JNH     no_rnd                          ; Round up if needed
        CMPB    AH,#0FFH              ; Don't increment if AH=0FFH
        JE      no_rnd
        INCB    AH

no_rnd: LDB     AL, AH                ; Align byte and change to word
        CLRB    AH
        ST      AX, ON_TIME[DX]

        INCB    AD_NUM
        ANDB    AD_NUM, #03H                    ; Keep AD_NUM between 0 and 3

next:   ADDB    AD_COMMAND, AD_NUM, #1000B      ; Start conversion on channel
                                                ; indicated by AD_NUM register
        ANDB    Port1, #11111011B            ; Clear P1.2
        POPF
        RET

        END
```

270658–37

Listing 4-1c. Interrupt Driven A to D Routine

The HSO routine shown in Listing 4-1b is slightly different than the one in section 3. All of the HSO lines turn on at the same time, only the turn-off-time is varied between lines. This action is what is most commonly required for multiple PWM outputs and simplifies the software. A comparison is made between Timer1 and the next HSO turn on time at the beginning of the routine. If the next turn on time has passed, then the on-times are loaded into the CAM, otherwise the off times are loaded.

The maximum number of events in the CAM at any given time is 7. This occurs when the first line to turn off does so, causing the off-times for all of the lines to be loaded. For two of the lines there will be an offtime, an on-time, and the just loaded off-time. The other line (the one that just turned off) will have only the on-time and the just loaded off-time.

A/D conversions are performed by the code in Listing 4-1c about every 60 microseconds, 42 for the conversion, the rest for overhead. The A/D routine sets up the HSO and PWM on and off times. Since the A/D has a ten bit output, the most significant 8 bits are rounded up or down based on the least significant two bits.

4.2. Software Serial Port Using the HSIO Unit

There are many systems which require more than one serial port, an example is a system which must communicate with other computers and have an additional port for a local console. If the on-board UART is being used as an inter-processor link, the HSIO unit can be used to interface the 8096 to an additional asynchronous line.

Figure 4-1 shows the format of a standard 10-bit asynchronous frame. The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the START bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are the eight data bits which are transmitted least significant bit first. The STOP bit is set to the opposite state of the START bit to guar-

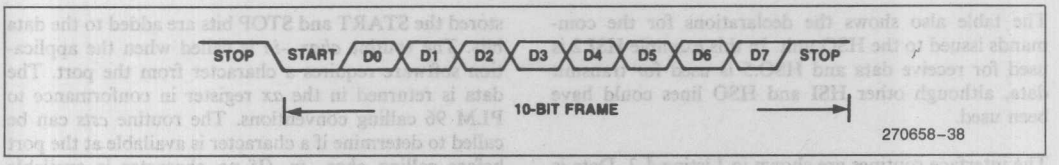


Figure 4-1. 10-bit Asynchronous Frame

antee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSIO unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.
3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt interrupt. Its function is to serialize the data passed to it by the interface routines.
4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial port.

```
;
;       VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
;       ============================================
;
        rseg

rcve_state:     dsb 1
  rxrdy         equ 1           ; indicates receive done
  rxoverrun     equ 2           ; indicates receive overflow
  rip           equ 4           ; receive in progress flag
rcve_buf:       dsb 1           ; used to double buffer receive data
rcve_reg:       dsb 1           ; used to deserialize receive
sample_time:    dsw 1           ; records last receive sample time

serial_out:     dsw 1           ; Holds the output character+framing (start and
                                ;   stop bits) for transmit process.
baud_count:     dsw 1           ; Holds the period of one bit in units
                                ;   of T1 ticks.
txd_time:       dsw 1           ; Transition time of last Txd bit that was
                                ;   sent to the CAM
char:           dsb 1           ; for test only
;
;       COMMANDS ISSUED TO THE HSO UNIT
;       ===============================
;
mark_command    equ     01101101b       ; timer1,set,interrupt on 5
space_command   equ     00101101b       ; timer1,clr,interrupt on 5
sample_command  equ     00110000b       ; software timer 0

$eject
```

270658-39

Listing 4-2. Software Serial Port Declarations

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char_out*, which waits for any in-process transmission to complete and stores the character into the variable *serial_out*. As the data is stored the START and STOP bits are added to the data bits. The routine *char—in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *csts* can be called to determine if a character is available at the port before calling *char_in*. (If no character is available *char_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```
;
char_out:
; Output character to the software serial port
;
        pop     cx                      ; the return address
        pop     bx                      ; the character for output
        ldb     (bx+1),#01h             ; add the start and stop bits
        add     bx,bx                   ;   to the char and leave as 16 bit
wait_for_xmit:
        cmp     serial_out,0            ; wait for serial_out=0 (it will be cleared by
        bne     wait_for_xmit           ;   the hso interrupt process)
        st      bx,serial_out           ; put the formatted character in serial_out
        br      [cx]                    ; return to caller
;
csts:
; Returns "true" (ax<>0) if char_in has a character.
;
        clr     ax
        bbc     rcve_state,0,csts_exit
        inc     ax
csts_exit:
        ret
;
char_in:
; Get a character from the software serial port
;
                                        ; wait for character ready
        bbc     rcve_state,0,char_in
        pushf                           ; set up a critical region
        andb    rcve_state,#not(rxrdy)
        ldbze   al,rcve_buf
        popf                            ; leave the critical region
        ret
```

270658-40

Listing 4-3. Software Serial Port Interface Routines

```
;
setup_serial_port:
; Called on system reset to intiate the software serial port.
;
        pop     cx                      ; the return address
        pop     bx                      ; the baud rate (in decimal)
        ld      dx,#0007h               ; dx:ax:=500,000 (assumes 12 Mhz crystal)
        ld      ax,#0A120h
        divu    ax,bx                   ; calculate the baud count (500,000/baudrate)
        st      ax,baud_count
        st      0,serial_out            ; clear serial out
        ldb     ioc1,#01100000b         ; Enable HSO.5 and Txd
        bbs     ios0,6,$                ; Wait for room in the HSO CAM
                                        ; and issue a MARK command.
        add     txd_time,timer1,20
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time
        clrb    rcve_buf                ; clear out the receive variables
        clrb    rcve_reg
        clrb    rcve_state
        call    init_receive            ; setup to detect a start bit
        br      [cx]                    ; return
```

270658-41

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial_out* register is zero a MARK (idle condition) is output. If the *serial_out* register contains data then the least significant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi_isr* and *software_timer_isr*. The listings of these routines are shown in Listings 4-6a,4-6b, and 4-6c respectively. The

```
;
hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
;       actual system.

        cseg    at 2006h
        dcw     hso_isr         ; Set up vector

        cseg
        pushf
        add     txd_time,baud_count
        cmp     serial_out,0    ; if character is done send a mark
        be      send_mark
        shr     serial_out,#1   ; else send bit 0 of serial_out and shift
        bc      send_mark       ;   serial_out left one place.
send_space:
        ldb     hso_command,#space_command
        ld      hso_time,txd_time
        br      hso_isr_exit
send_mark:
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time

hso_isr_exit:
        popf
        ret
$eject
```

270658-42

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process

```
;
init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
        ldb     ioc0,#00000000b         ; disconnect change detector
        ldb     hsi_mode,#00100000b     ; negative edges on HSI.2
flush_fifo:
        orb     iosl_save,iosl
        bbc     iosl_save,7,flush_fifo_done
        ldb     al,hsi_status
        ld      ax,hsi_time             ; trash the fifo entry
        andb    iosl_save,#not(80h)     ; clear bit 7.
        br      flush_fifo
flush_fifo_done:
        ldb     ioc0,#00010000b         ; connect HSI.2 to detector
        ret
```

270658-43

Listing 4-6a. Software Serial Port Receive Initialization

```
;
hsi_isr:
; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
        cseg at 2004h
        dcw     hsi_isr                 ; setup the interrupt vector

        cseg
        pushf
        push    ax
        ldb     al,hsi_status
        ld      sample_time,hsi_time
        bbc     al,4,exit_hsi
        bbs     ios0,7,$                ; wait for room in HSO holding reg
        ld      ax,baud_count           ; send out sample command in 1/2
        shr     ax,#1                   ; bit time
        add     sample_time,ax
        ldb     hso_command,#sample_command
        st      sample_time,hso_time
        ldb     ioc0,#00000000b         ; disconnect hsi.2 from change detector
exit_hsi:
        pop     ax
        popf
        ret
```

270658-44

Listing 4-6b. Software Serial Port Start Bit Detect

```
;
software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incomming data.
; Note: this routine would be incorporated into the software timer stategy
; in an actual system.
;
        cseg at 200ah
        dcw     software_timer_isr      ; setup vector

        cseg
        pushf
        orb     iosl_save,iosl
        andb    iosl_save,#not(01h)     ; clear bit 0
        andb    0,rcve_state,#0fch      ; All bits except rxrdy and overrun=0
        bne     process_data
process_start_bit:
        bbc     hsi_status,5,start_ok
        call    init_receive
        br      software_timer_exit
start_ok:
        orb     rcve_state,#rip ; set receive in progress flag
        br      schedule_sample

process_data:
        bbs     rcve_state,7,check_stopbit
        shrb    rcve_reg,#1
        bbc     hsi_status,5,datazero
        orb     rcve_reg,#80h   ; set the new data bit
datazero:
        addb    rcve_state,#10h ; increment bit count
        br      schedule_sample

check_stopbit:
        bbc     hsi_status,5,$  ; DEBUG ONLY
        ldb     rcve_buf,rcve_reg
        orb     rcve_state,#rxrdy
        andb    rcve_state,#03h ; Clear all but ready and overrun bits
        call    init_receive
        br      software_timer_exit

schedule_sample:
        bbs     ios0,7,$        ; wait for holding reg empty
        ldb     hso_command,#sample_command
        add     sample_time,baud_count
        st      sample_time,hso_time

software_timer_exit:
        popf
        ret
```

270658-45

Listing 4-6c. Software Serial Port Data Reception

start is detected by the *hsi_isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve_state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init_receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

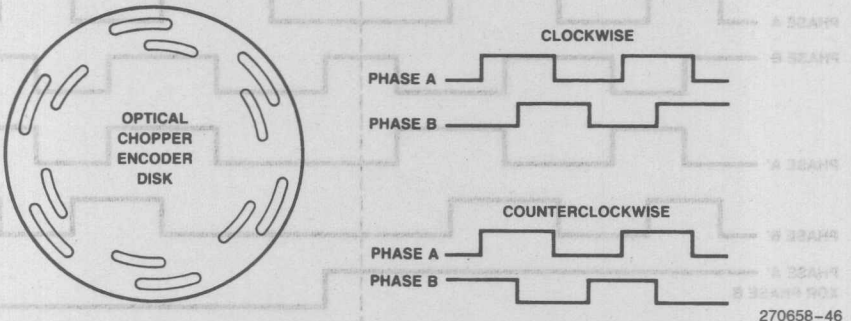
4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.



Inside track generates Phase A. Outside track generates Phase B.

Figure 4-2. Optical Encoder and Waveforms

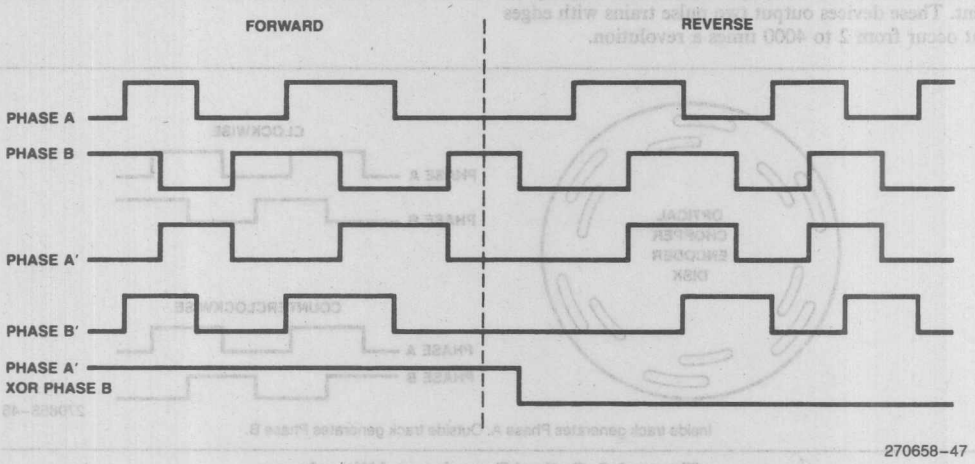
Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ±15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with



NOTES:
Phase A' is Phase A clocked by Phase B
Phase B' is Phase B clocked by Phase A

Figure 4-3. Filtered Encoder Waveforms

some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

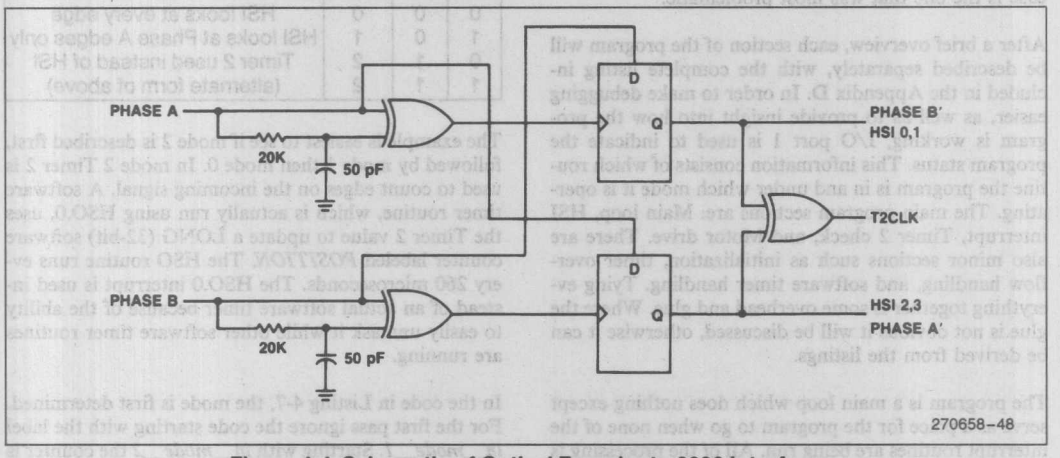


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

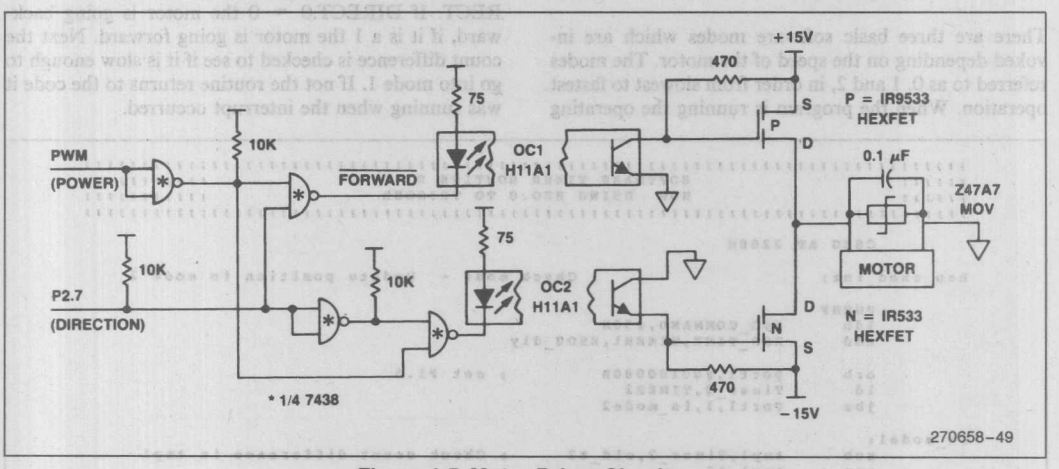


Figure 4-5. Motor Driver Circuitry

motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the operating mode is indicated by the lower 2 bits of Port 1, with the following coding:

P1.0	P1.1	Mode	Description
0	0	0	HSI looks at every edge
1	0	1	HSI looks at Phase A edges only
0	1	2	Timer 2 used instead of HSI
1	1	2	(alternate form of above)

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in_mode_1*. Starting with *in_mode_2* the counter is incremented or decremented based on bit zero of DIRECT. If DIRECT.0 = 0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                    SOFTWARE TIMER ROUTINE 0                 ;;;;;;;;;;;
;;;;;;                    NOW  USING HSO.0 TO TRIGGER              ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2280H

hso_exec_int:                     ; Check mode -  Update position in mode 2

        PUSHF
        ldb     HSO_COMMAND,#30H
        add     HSO_TIME,TIMER1,HSO0_dly

        orb     port1,#00100000B          ; set P1.5
        ld      Timer_2,TIMER2
        jbs     Port1,1,in_mode2

in_mode1:
        sub     tmp1,Timer_2,old_t2       ; Check count difference in tmp1
        cmp     tmp1,#2
        jh      end_swt0
set_mode0:
        jbc     Port1,0,end_swt0          ; if already in mode 0
        andb    Port1,#11111100B          ; Clear P1.0, P1.1 (set mode 0)
        ldb     IOC0,#01010101B           ; enable all HSI
        ldb     last_stat,zero
        br      end_swt0
```

270658–50

Listing 4-7. Motor Control HSO.0 Timer Routine

```
in_mode2:
        sub     delta_p,timer_2,tmr2_old        ; get timer2 count difference
        ld      tmr2_old,timer_2

        jbc     direct,0,in_rev

in_fwd: add     position,delta_p
        addc    position+2,zero
        br      chk_mode

in_rev: sub     position,delta_p
        subc    position+2,zero

chk_mode:
        sub     tmp1,Timer_2,old_t2     ; Check count difference in tmp1
        cmp     tmp1,#5                 ; set model if count is too low
        jgt     end_swt0                ; count <= 5

set_model:
        andb    Port1,#11111101B        ; Clear P1.1, set P1.0 (set mode 1)
        orb     Port1,#00000001B
        ldb     IOC0,#00000101B         ; enable HSI 0 and 1
        ld      zero, HSI_TIME
        sub     last1_time,Timer1,min_hsi1
                ; set up so (time-last2_time)>min_hsi1 on next HSI
clr_hsi:
        ld      ZERO, HSI_TIME
        andb    iosl_bak,#01111111B             ; clear bit 7
        orb     iosl_bak,iosl
        jbs     iosl_bak,7,clr_hsi      ; If hsi is triggered then clear hsi

end_swt0:
        ld      old_t2,TIMER_2
        andb    port1,#11011111B        ; clear P1.5
        POPF
        ret
```

270658-51

Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI_TIME* register is read to speed up clearing the HSI FIFO and the *LAST1_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

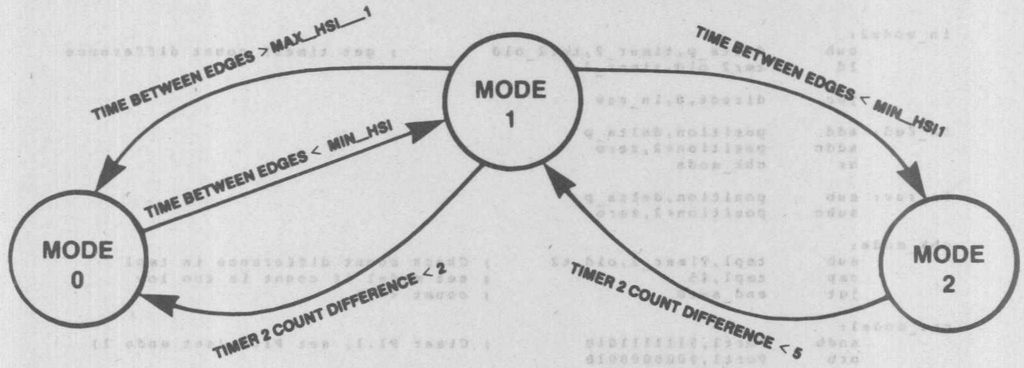
This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in_mode1* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is run. At the end time and count values are saved and the



270658-52

NOTES:
Mode 0: HSI Examines edges on Phase A and B
Mode 1: HSI Examines edges on Phase A only
Mode 2: TIMER 2 stores edgecount

Figure 4-6. Mode State Diagram

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;          HSI DATA AVAILABLE INTERRUPT ROUTINE               ;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;   This routine keeps track of the current time and position of the motor.
;   The upper word of information is provided by the timer overflow routine.

        CSEG AT 2400H
now_mode_1:     br      in_mode_1               ; used to save execution time for
no_intl:        br      no_int                  ; worst case loop

hsi_data_int:   pushf
        orb     port1,#01000000B                ; set P1.6
        andb    iosl_bak,#01111111B             ; Clear iosl_bak.7
        orb     iosl_bak,iosl
        jbc     iosl_bak,7,no_intl              ; If hsi is not triggered then
                                                ; jump to no_int
get_values:
        ld      timer_2,TIMER2
        andb    hsi_s0,HSI_STATUS,#01010101B
        ld      time, HSI_TIME

        jbs     port1,0,now_mode_1              ; jump if in mode 1

In_mode_0:

;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;               INSERT    BODY    OF    ROUTINE
;;;;;;;;;;;;;;;;;;;

load_lasts:
        ld      tmr2_old,timer_2
no_cnt: andb    iosl_bak,#01111111B             ; clr bit 7
        orb     iosl_bak,iosl
        jbc     iosl_bak,7,no_int
again:  br      get_values

no_int: andb    port1,#10111111B                ; Clear P1.6
        popf
        ret     ; end of hsi_data interrupt routine
                ; Routine for mode 1 follows and then returns to "load_lasts"
$EJECT
```

270658-53

Listing 4-8. Motor Control HSI Data Available Routine

holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. LAST_STAT = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update *POSITION*. The program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

```
In_mode_1:                  ; mode 1 HSI routine

          andb      tmp1,hsi_s0,#01010000B
          jne       no_cnt
cmp_time:                                   ; Procedure which sets mode 1 also
                                            ; sets times to pass the tests
          ld        last2_time,last1_time
          ld        last1_time,time

cmp1:     sub       tmp1,time,last2_time
          cmp       tmp1,min_hsi1
          jh        check_max_time

set_mode_2:
          orb       Port1,#00000010B        ; Set P1.1 (in mode 2)
          ldb       IOC0,#00000000B         ; Disable all HSI
mt_hsi:   ld        zero,hsi_time           ; empty the hsi fifo
          andb      iosl_bak,#01111111B               ; clear bit 7
          orb       iosl_bak,iosl
          jbs       iosl_bak,7,mt_hsi       ; If hsi is triggered then clear hsi
          br        done_chk

check_max_time:
          sub       tmp1,time,last2_time
          cmp       tmp1,max_hsi1           ; max_hsi = addition to min_hsi for
                                            ; total time
          jnh       done_chk

set_mode_0:
          andb      Port1,#11111100B        ; clear P1.0,1  set mode 0)
          ldb       IOC0,#01010101B         ; Enable all HSI
          ldb       last_stat,zero

done_chk:
          sub       delta_p,timer_2,tmr2_old          ; get timer2 count difference
          jbc       direct,0,add_rev
add_fwd:
          add       position,delta_p
          addc      position+2,zero
          br        load_lasts
add_rev:
          sub       position,delta_p
          subc      position+2,zero
          br        load_lasts

$eject
```

270658-54

Listing 4-9. Motor Control Mode 1 Routines

```
In_mode_0:
          jbs     hsi_s0,0,a_rise
          jbs     hsi_s0,2,a_fall
          jbs     hsi_s0,4,b_rise
          jbs     hsi_s0,6,b_fall
          br      no_cnt

a_rise:   ld      last2_time,last1_time
          ld      last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh      tst_statr
;set model-
          orb     Port1,#00000001B          ; Set P1.0 (in mode 1)
          ldb     IOC0,#00000101B           ; Enable HSI 0 and 1
tst_statr:
          jbs     last_stat,6,going_fwd
          jbs     last_stat,4,going_rev
          jbs     last_stat,2,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

a_fall:   ld      last2_time,last1_time
          ld      last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh      tst_statf
;set model-
          orb     Port1,#00000001B          ; Set P1.0 (in mode 1)
          ldb     IOC0,#00000101B           ; Enable HSI 0 and 1
tst_statf:
          jbs     last_stat,4,going_fwd
          jbs     last_stat,6,going_rev
          jbs     last_stat,0,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

b_rise:   jbs     last_stat,0,going_fwd
          jbs     last_stat,2,going_rev
          jbs     last_stat,6,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

b_fall:   jbs     last_stat,2,going_fwd
          jbs     last_stat,0,going_rev
          jbs     last_stat,4,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

first_time:
          stb     hsi_s0,last_stat
          br      done_chk          ; add delta position
inp_err:
          br      no_int

change_dir:
          notb    direct
no_inc:   jbc     direct,0,going_rev

going_fwd:
          orb     PORT2,#01000000B          ; set P2.6
          ldb     direct,#01                ; direction = forward
          add     position,#01
          addc    position+2,zero
          br      st_stat
going_rev:
          andb    PORT2,#10111111B          ; clear P2.6
          ldb     direct,#00                ; direction = reverse
          sub     position,#01
          subc    position+2,zero

st_stat:
          stb     hsi_s0,last_stat
```

270658-55

Listing 4-10. Motor Control Mode 0 Routines

After mode correctness is confirmed and the *LASTx_TIME* values are updated the *LAST_STAT* (Last Status) variable is used to determine the current direction of travel. The POSITION value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done_chk* routine is used to update POSITION, instead of the routines *going_fwd* and *going_rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and timer errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to see if the motor is close enough to the desired location that the power to it should be reversed, (i.e., enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold_Position mode is entered.*

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold_Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The BOOST variable provides this integral control by increasing the output a fixed amount every time period in which the

Listing 4-11. Motor Control Software Timer 1 Routine

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                     SOFTWARE TIMER ROUTINE 1                  ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CSEG AT 2600H

swtl_expired:

        pushf
        orb     portl,#10000000B        ; set portl.7

        ldb     int_mask,#00001101B     ; enable HSI, Tovf, HSO

        ldb     HSO_COMMAND,#39H
        add     HSO_TIME,TIMER1,swtl_dly

        ld      time_err+2,des_time+2   ; Calculate time & position error
        ld      pos_err+2,des_pos+2
        sub     time_err,des_time,time         ; values are set
        subc    time_err+2,time+2
        sub     pos_err,des_pos,position
        subc    pos_err+2,position+2

        EI

        sub     time_delta,last_time_err,time_err
        ld      last_time_err,time_err

        sub     pos_delta,last_pos_err,pos_err
        ld      last_pos_err,pos_err

;;;;;           Time_err  = Desired time to finish - current time
;;;;;           Pos_err   = Desired position to finish - current position
;;;;;           Pos_delta  = Last position error - Curent position error
;;;;;           Time_delta = Last time error - Current time error
;;;;;              note that errors should get smaller so deltas will be
;;;;;              positive for forward motion (time is always forward)
```

270658-56

Listing 4-11a. Motor Control Software Position Counter

```
chk_dir:
        cmp     pos_err+2,zero
        jge     go_forward

go_backward:
        neg     pos_err             ; Pos_err = ABS VAL (pos_err)
        ldb     pwm_dir,#00h
        cmp     pos_err+2,#0ffffH
        jne     ld_max
        br      chk_brk

go_forward:
        ldb     pwm_dir,#01H
        cmp     pos_err+2,zero
        je      chk_brk

ld_max: ldb     pwm_pwr,max_pwr
        br      chk_sanity

Chk_brk:                            ;  Position_Error now = ABS(pos_err)
        cmp     pos_err,pos_pnt
        jnh     hold_position       ; position_error<position_control_point
        cmp     pos_err,brk_pnt
        jh      ld_max              ; position_error>brake_point

braking:
        cmp     pos_delta,zero
        jge     chk_delta
        neg     pos_delta
chk_delta:
        cmp     pos_delta,vel_pnt           ; velocity = pos_delta/sample_time
        jnh     hold_position               ; jmp if  ABS(velocity) < vel_pnt

brake:  ldb     pwm_pwr,max_brk
        ldb     tmp,direct                  ; If braking apply power in opposite
        notb    tmp                         ; direction of current motion
        ldb     pwm_dir,tmp

        br      ld_pwr

Hold_position:                      ; position hold mode
        cmp     pos_err,#02
        jh      calc_out            ; if position error < 2 then turn off power
        clr     tmp+2
        clr     boost
        BR      output

calc_out:
        mulub   tmp,max_hold,#255
        mulu    tmp,pos_err                 ; Tmp = pos_err * max_hold
        cmp     pos_delta,zero
        jne     no_bst
        add     boost,#04                   ; Boost is integral control
        add     tmp+2,boost                 ; TMP+2 = MSB(pos_err*max_hold)
        br      ck_max
no_bst: clr     boost
ck_max: cmp     tmp+2,max_hold
        jnh     output
maxed:  ld      tmp+2,max_hold
output: ldb     pwm_pwr,tmp+2

chk_sanity:
        br      ld_pwr

ld_pwr:
        ldb     rpwr,pwm_pwr
        notb    rpwr
        jbs     pwm_dir,0,p2fwd

p2bkwd: DI
        andb    port2,#01111111B            ; clear P2.7
        ldb     pwm_control,rpwr
        EI
        br      pwrset
p2fwd:  DI
        orb     port2,#10000000B            ; set P2.7
        ldb     pwm_control,rpwr
        EI
```

270658-57

Listing 4-11b: Motor Control Power Algorithm

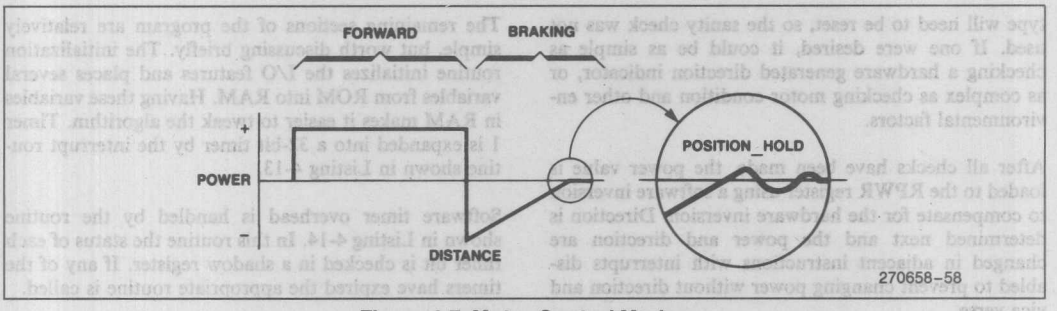


Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-

```
pwrset:
        cmp     time_err+2,zero  ; do pos_table when err is negative
        jgt     end_p
;;;     br      end_p

        cmp     nxt_pos,#(32+pos_table)
        jlt     get_vals                ; jump if lower
        ld      nxt_pos,#pos_table
        clr     time+2
get_vals:

        ld      des_pos,[nxt_pos]+
        ld      des_pos+2,[nxt_pos]+
        ld      des_time+2,[nxt_pos]+
        ld      max_pwr,[nxt_pos]+
        ld      max_brk,max_pwr
        add     des_pos,offset
        addc    des_pos+2,zero
        sub     last_pos_err,des_pos,position

end_p:  andb    portl,#01111111B        ; clear P1.7

        popf
        ret

pos_table:

        dcl     00000000H       ; position 0
        dcw     0020H, 0080H    ; next time, power
        dcl     0000c000H       ; position 1
        dcw     0040H, 0040H    ; next time, power
        dcl     00000000H       ; position 2
        dcw     0060H, 00c0H    ; next time, power
        dcl     0FFFF8000H      ; position 3
        dcw     0080H, 0080H    ; next time, power

        dcl     00000800H       ; position 4
        dcw     0058H, 0080H    ; next time, power
        dcl     00003000H       ; position 5
        dcw     0070H, 00ffH    ; next time, power
        dcl     00000000H       ; position 6
        dcw     0090H, 00f0H    ; next time, power
        dcl     00000000H       ; position 7
        dcw     0091H, 00f0H    ; next time, power
```

270658-59

Listing 4-12. Motor Control Next Position Lookup

type will need to be reset, so the sanity check was not used. If one were desired, it could be as simple as checking a hardware generated direction indicator, or as complex as checking motor condition and other environmental factors.

After all checks have been made, the power value is loaded to the RPWR register using a software inversion to compensate for the hardware inversion. Direction is determined next and the power and direction are changed in adjacent instructions with interrupts disabled to prevent changing power without direction and vice versa.

To exercise the program logic the desired position is changed based on the time value using the code and lookup table shown in Listing 4-12.

The remaining sections of the program are relatively simple, but worth discussing briefly. The initialization routine initializes the I/O features and places several variables from ROM into RAM. Having these variables in RAM makes it easier to tweak the algorithm. Timer 1 is expanded into a 32-bit timer by the interrupt routine shown in Listing 4-13.

Software timer overhead is handled by the routine shown in Listing 4-14. In this routine the status of each timer bit is checked in a shadow register. If any of the timers have expired the appropriate routine is called.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                   TIMER  INTERRUPT  SERVICE                    ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2200H

timer_ovf_int:
        pushf

        orb     iosl_bak,IOS1
chk_tl: jbc     iosl_bak,5,tmr_int_done
        inc     time+2
        andb    iosl_bak,#11011111B     ; clear bit 5
tmr_int_done:
        popf
        ret                 ; End of timer interrupt routine
```

270658–60

Listing 4-13. Motor Control Timer Interrupt Routine

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;           SOFTWARE TIMER INTERRUPT SERVICE ROUTINE             ;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2220H


soft_tmr_int:
        pushf
        orb     iosl_bak,IOS1
chk_swt0:
        jbc     iosl_bak,0,chk_swt1
        andb    iosl_bak,#11111110B     ; Clear bit 0 - end swt0
        call    swt0_expired
chk_swt1:
        jbc     iosl_bak,1,chk_swt2
        andb    iosl_bak,#11111101B     ; Clear bit 1
        call    swt1_expired
chk_swt2:
        jbc     iosl_bak,2,chk_swt3
        andb    iosl_bak,#11111011B     ; Clear bit 2
        call    swt2_expired
chk_swt3:
        jbc     iosl_bak,4,swt_int_done
        andb    iosl_bak,#11110111B     ; Clear bit 3
;       call    swt3_expired

swt_int_done:
        popf
        ret     ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

$eject
```

270658–B2

Listing 4-14. Motor Control Software Timer Interrupt Handler

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                 SOFTWARE TIMER ROUTINE 2                    ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2380H

swt2_expired:
        pushf
        ldb     hso_command,#3AH            ; set swt_2
        add     hso_time,timer1,swt2_dly

        orb     port1,#00000100B            ; set port 1.2
        cmp     out_ptr,#7ffH
        bnh     pulsing
        ld      out_ptr,#1f0H

pulsing:
        jbc     tr_col,0,swt2_done

        st      position+2,[out_ptr]+       ; position high, position low
        st      position,[out_ptr]+

        st      direct,[out_ptr]+
        st      pwm_pwr,[out_ptr]+
                                            ; store 8 bytes externally

swt2_done:
        sub     tmp1,timer1,last1_time
        cmp     tmp1,#1800H
        jnh     swt2_ret            ; keep (time_last4_time)<7000H

        add     last1_time,#1000H
swt2_ret:
        andb    port1,#11111011B            ; clear port1.2
        popf
        ret
```

270658-61

Listing 4-15: Motor Control Software Timer 2 Routine

The last routine, shown in Listing 4-15, is the Software Timer 2 routine which outputs some variables to external RAM. It also keeps LAST1_Time within 1800H of Timer1 to prevent overflows from occurring when the Mode 0 and Mode 1 software check this variable.

A complete listing of the program as it is used in our lab can be found in Appendix D. For a given motor or encoder it will probably be necessary to change some of the time constants on the first page of the listing. With the motor used in our experimentation, pulses are missed from time to time when direction changes quickly. If the motor were not as fast to turn around or the encoder were mounted better these problems should disappear. The missing pulses occur when switching from Mode 1 to Mode 0, other than that no anomalies were found in the lab.

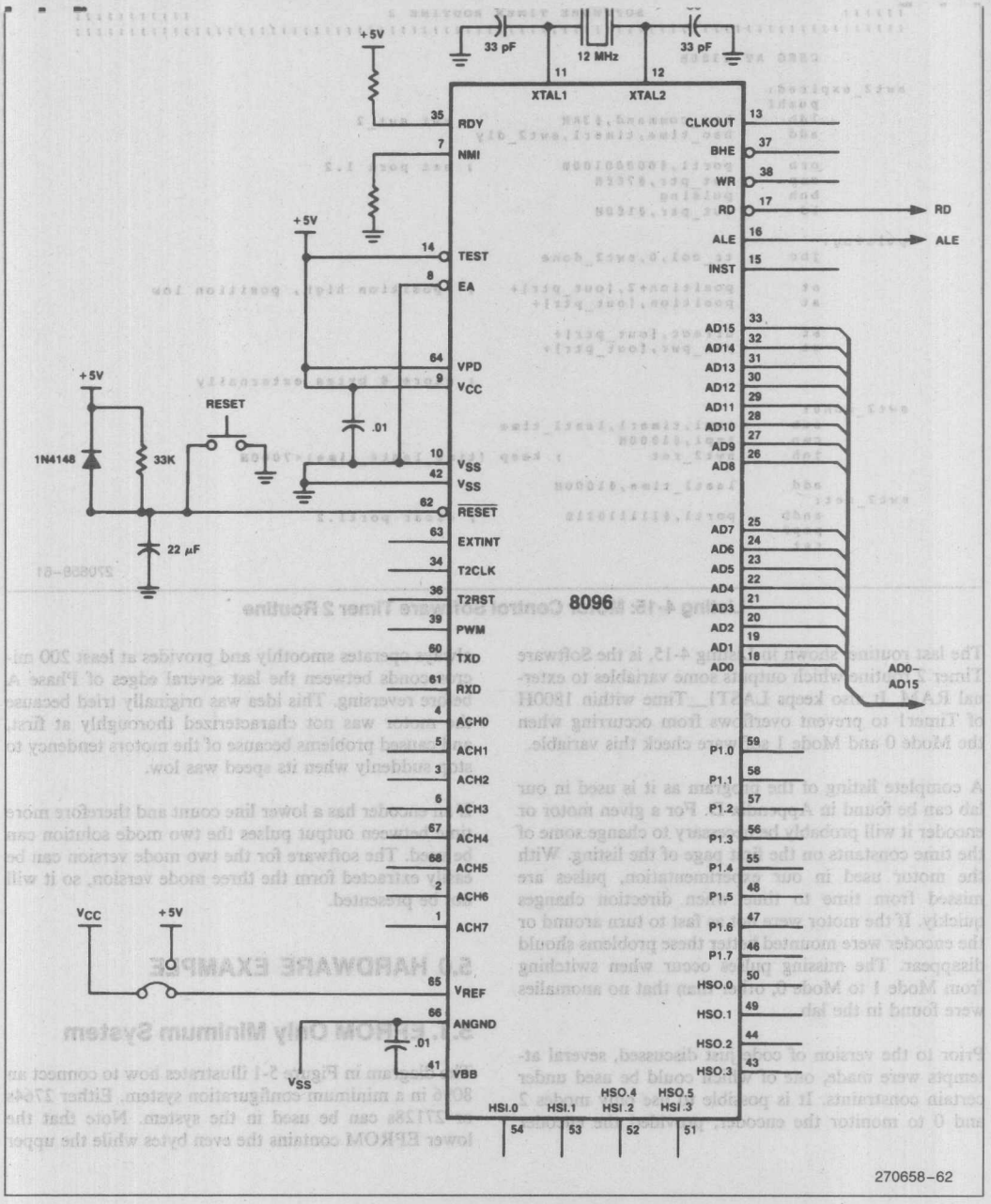
Prior to the version of code just discussed, several attempts were made, one of which could be used under certain constraints. It is possible to use only modes 2 and 0 to monitor the encoder, provided the encoder always operates smoothly and provides at least 200 microseconds between the last several edges of Phase A before reversing. This idea was originally tried because the motor was not characterized thoroughly at first, and caused problems because of the motors tendency to stop suddenly when its speed was low.

If an encoder has a lower line count and therefore more time between output pulses the two mode solution can be used. The software for the two mode version can be easily extracted form the three mode version, so it will not be presented.

5.0 HARDWARE EXAMPLE

5.1. EPROM Only Minimum System

The diagram in Figure 5-1 illustrates how to connect an 8096 in a minimum configuration system. Either 2764s or 27128s can be used in the system. Note that the lower EPROM contains the even bytes while the upper



270658-62

one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper 4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

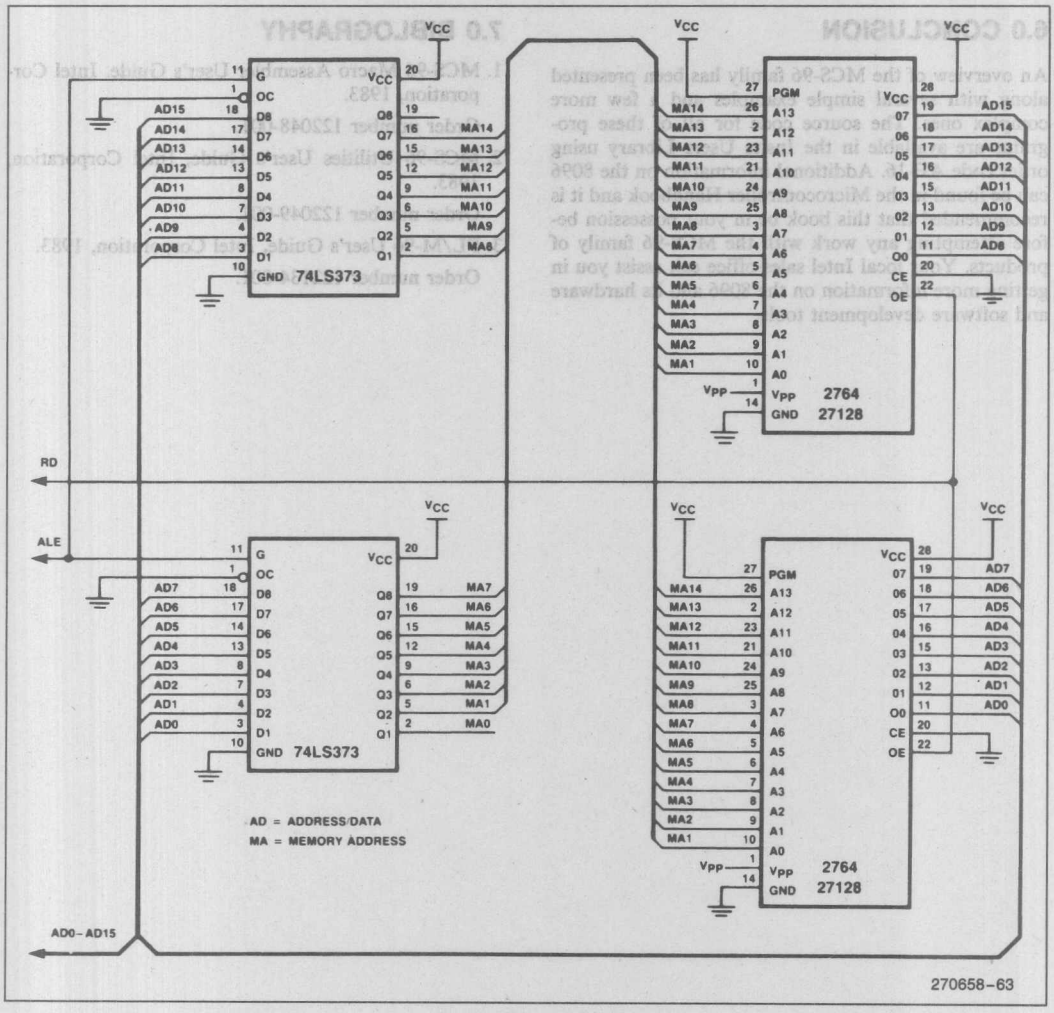


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

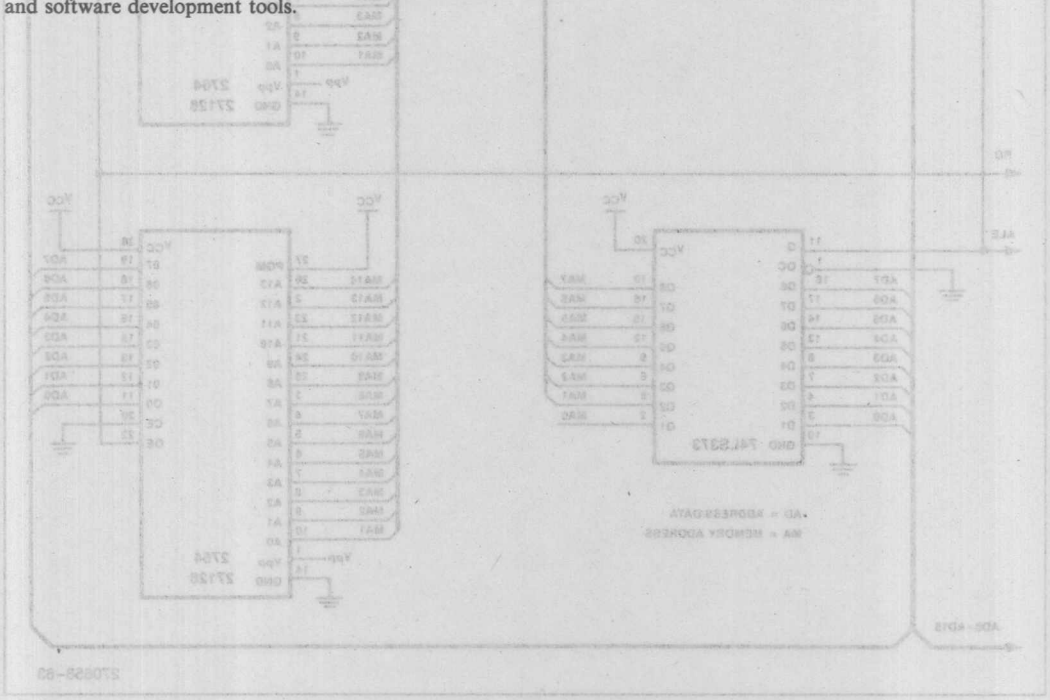
The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Handbook and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

7.0 BIBLOGRAPHY

1. MCS-96 Macro Assembler User's Guide, Intel Corporation, 1983.
   Order number 122048-001.
2. MCS-96 Utilities User's Guide, Intel Corporation, 1983.
   Order number 122049-001.
3. PL/M-96 User's Guide, Intel Corporation, 1983.
   Order number 122134-001.



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:INTER1.A96
OBJECT FILE:   F3:INTER1.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE      SOURCE STATEMENT
                                    1      $TITLE('INTER1.A96:  Interpolation routine 1')
                                    2      ;;;;;;;   8096 Assembly code for table lookup and interpolation
                                    3
                                    4      $INCLUDE(:F0:DEMO96.INC)      ; Include demo definitions
                               =1   5      $nolist  ;  Turn listing off for include file
                               =1  53              ;  End of include file
                                   54
     0022                          55      RSEG  at 22H
                                   56
     0022                          57              IN_VAL:          dsb     1              ; Actual Input Value
     0024                          58              TABLE_LOW:       dsw     1
     0026                          59              TABLE_HIGH:      dsw     1
     0028                          60              IN_DIF:          dsw     1              ; Upper Input - Lower Input
       0028                        61              IN_DIFB          equ     IN_DIF  :byte
     002A                          62              TAB_DIF:         dsw     1              ; Upper Output - Lower Output
     002C                          63              OUT:             dsw     1
     002E                          64              RESULT:          dsw     1
     0030                          65              OUT_DIF:         dsl     1              ; Delta Out
                                   66
                                   67
     2080                          68      CSEG at 2080H
                                   69
     2080 A1000118                 70              LD      SP, #100H
                                   71
     2084 B0221C                   72      look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
     2087 18031C                   73              SHRB    AL, #3          ; Divide the byte by 8
     208A 71FE1C                   74              ANDB    AL, #11111110B ; Insure AL is a word address
                                   75                                      ;  This effectively divides AL by 2
                                   76                                      ;  so AL = IN_VAL/16
                                   77
     208D AC1C1C                   78              LDBZE   AX, AL          ; Load byte AL to word AX
     2090 A31D002124               79              LD      TABLE_LOW, TABLE [AX]    ; TABLE_LOW is loaded with the value
                                   80                                               ;  in the table at table location AX
                                   81
```

270658-64

```
2095 A31D022126          82          LD      TABLE_HIGH, (TABLE+2)[AX]  ; TABLE_HIGH is loaded with the
                         83                                             ; value in the table at table
                         84                                             ; location AX+2
                         85                                             ; (The next value in the table)
                         86
209A 4824262A            87          SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                         88                                             ; TAB_DIF=TABLE_HIGH-TABLE_LOW
                         89
209E 510F2228            90          ANDB    IN_DIFB, IN_VAL, #0FH      ; IN_DIFB=least significant 4 bits
                         91                                             ;  of IN_VAL
20A2 AC2828              92          LDBZE   IN_DIF, IN_DIFB            ; Load byte IN_DIFB to word IN_DIF
                         93
20A5 FE4C2A2830          94          MUL     OUT_DIF, IN_DIF, TAB_DIF
                         95                                             ; Output_difference =
                         96                                             ;  Input_difference*Table_difference
20AA 0E0430              97          SHRAL   OUT_DIF, #4                ; Divide by 16 (2**4)
                         98
20AD 4424302C            99          ADD     OUT, OUT_DIF, TABLE_LOW    ; Add output difference to output
                        100                                             ;  generated with truncated IN_VAL
                        101                                             ;  as input
20B1 0A042C             102          SHRA    OUT, #4                    ; Round to 12-bit answer
20B4 A4002C             103          ADDC    OUT, zero                  ; Round up if Carry = 1
                        104
20B7 C02E2C             105  no_inc: ST      OUT, RESULT                ; Store OUT to RESULT
                        106
20BA 27C8               107          BR      look                       ; Branch to "look:"
                        108
                        109
2100                    110  cseg    AT 2100H
                        111
2100 0000002000034004C  112  table:  DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078   113          DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D   114          DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022   115          DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010               116          DCW     1000H
                        117
2122                    118  END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-65

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:INTER2.A96
OBJECT FILE:   F3:INTER2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('INTER2 A96:  Interpolation routine 2')
                                    2
                                    3   ;;;;;;;    8096 Assembly code for table lookup and interpolation
                                    4   ;;;;;;;    Using tabled values in place of division
                                    5
                                    6   $INCLUDE(:F0:DEMO96.INC) ; Include demo definitions
                               =1   7   $nolist  ;  Turn listing off for include file
                               =1  55            ;  End of include file
                                   56
    0024                           57   RSEG  at 24H
                                   58
    0024                           59        IN_VAL:          dsb     1            ; Actual Input Value
    0026                           60        TABLE_LOW:       dsw     1            ; Table value for function
    0028                           61        TABLE_INC:       dsw     1            ; Incremental change in function
    002A                           62        IN_DIF:          dsw     1            ; Upper Input - Lower Input
      002A                         63        IN_DIFB          equ     IN_DIF  :byte
    002C                           64        OUT:             dsw     1
    002E                           65        RESULT:          dsw     1
    0030                           66        OUT_DIF:         dsl     1            ; Delta Out
                                   67
                                   68
    2080                           69   CSEG at 2080H
                                   70
    2080 A1000118                  71        LD      SP, #100H       ; Initialize SP to top of reg. file
                                   72
    2084 B0241C                    73   look:LDB     AL, IN_VAL      ; Load temp with Actual Value
    2087 18031C                    74        SHRB    AL, #3          ; Divide the byte by 8
    208A 71FE1C                    75        ANDB    AL, #11111110B  ; Insure AL is a word address
                                   76                                ;  This effectively divides AL by 2
                                   77                                ;  so AL = IN_VAL/16
    208D AC1C1C                    78        LDBZE   AX, AL          ; Load byte AL to word AX
                                   79
    2090 A31D002126                80        LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                   81                                 ;  in the value table at location AX
                                   82
    2095 A31D222128                83        LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                   84                                 ; in the increment table at
                                   85                                 ; location AX+2
                                   86
```

270658-66

```
209A 510F242A          87              ANDB    IN_DIFB, IN_VAL, #0FH   ; IN_DIFB=least significant 4 bits
                       88                                              ;  of IN_VAL
209E AC2A2A            89              LDBZE   IN_DIF, IN_DIFB         ; Load byte IN_DIFB to word IN_DIF
                       90
20A1 FE4C282A30        91              MUL     OUT_DIF, IN_DIF, TABLE_INC
                       92                                              ; Output_difference =
                       93                                              ;  Input_difference*Incremental_change
                       94
20A6 4426302C          95              ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                       96                                              ;  generated with truncated IN_VAL
                       97                                              ;  as input
20AA 08042C            98              SHR     OUT, #4                 ; Round to 12-bit answer
20AD A4002C            99              ADDC    OUT, zero               ; Round up if Carry = 1
                      100
20B0 C02E2C           101  no_inc:     ST      OUT, RESULT             ; Store OUT to RESULT
20B3 27CF             102              BR      look                    ; Branch to "look:"
                      103
                      104
2100                  105  cseg        AT 2100H
                      106
2100                  107  val_table:
2100 0000002000034004C 108              DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078 109              DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D 110              DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022 111              DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010             112              DCW     1000H
2122                  113  inc_table:
2122 0002400180011001 114              DCW      0200H,  0140H,  0180H,  0110H   ; Table of incremental
212A D000800060003000 115              DCW      00D0H,  0080H,  0060H,  0030H   ; differences
2132 200090FF70FF00FF 116              DCW     0020H, 0FF90H, 0FF70H, 0FF00H
213A E0FE90FEE0FEE0FE 117              DCW     0FEE0H, 0FE90H, 0FEE0H, 0FEE0H
                      118
2142                  119  END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-67

```
SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE PLMEX
OBJECT MODULE PLACED IN :F3:PLMEX1.OBJ
COMPILER INVOKED BY:  PLM96.86  F3:PLMEX1.P96 CODE


          $TITLE('PLMEX1:  PLM-96 Example Code for Table Lookup')

          /* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

   1      PLMEX:    DO;

   2   1  DECLARE IN_VAL          WORD      PUBLIC;
   3   1  DECLARE TABLE_LOW      INTEGER   PUBLIC;
   4   1  DECLARE TABLE_HIGH     INTEGER   PUBLIC;
   5   1  DECLARE TABLE_DIF      INTEGER   PUBLIC;
   6   1  DECLARE OUT            INTEGER   PUBLIC;
   7   1  DECLARE RESULT         INTEGER   PUBLIC;
   8   1  DECLARE OUT_DIF        LONGINT   PUBLIC;
   9   1  DECLARE TEMP           WORD      PUBLIC;

  10   1  DECLARE TABLE(17)      INTEGER  DATA (
                  0000H, 2000H, 3400H, 4C00H,      /* A random function */
                  5D00H, 6A00H, 7200H, 7800H,
                  7B00H, 7D00H, 7600H, 6D00H,
                  5D00H, 4B00H, 3400H, 2200H,
                  1000H);

  11   1  DMPY:  PROCEDURE (A,B) LONGINT EXTERNAL;
  12   2          DECLARE (A,B) INTEGER;
  13   2  END DMPY;

  14   1  LOOP
            TEMP=SHR(IN_VAL,4);      /* TEMP is the most significant 4 bits of IN_VAL */

  15   1        TABLE_LOW=TABLE(TEMP);      /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
  16   1        TABLE_HIGH=TABLE(TEMP+1);   /* The code would work but the 8096 would     */
                                            /* do two shifts                              */

  17   1        TABLE_DIF=TABLE_HIGH-TABLE_LOW;

  18   1        OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND OFH)) /16;

  19   1        OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                                   in this case 4 places are shifted     */
```

270658-68

```
20   1          IF CARRY=0 THEN RESULT=OUT;   /* Using the hardware flags must be done   */
22   1            ELSE  RESULT=OUT+1;         /* with care to ensure the flag is tested  */
                                               /* in the desired instruction sequence     */
23   1      GOTO LOOP;

           /* END OF PLM-96 CODE  */

24   1      END;
```

270658-69

```
PL/M-96 COMPILER    PLMEX1:  PLM-96 Example Code for Table Lookup
                    ASSEMBLY LISTING OF OBJECT CODE

                                          ;   STATEMENT     14
         0022                       PLMEX:
         0022  A1000018          R        LD     SP,#STACK
         0026                       LOOP:
         0026  A00010             R        LD     TEMP,IN_VAL
         0029  0B0410             R        SHR    TEMP,#4H
                                          ;   STATEMENT     15
         002C  4410101C           R        ADD    TMP0,TEMP,TEMP
         0030  A31D000002         R        LD     TABLE_LOW,TABLE[TMP0]
                                          ;   STATEMENT     16
         0035  A31D020004         R        LD     TABLE_HIGH,TABLE+2H[TMP0]
                                          ;   STATEMENT     17
         003A  48020406           R        SUB    TABLE_DIF,TABLE_HIGH,TABLE_LOW
                                          ;   STATEMENT     18
         003E  C806               R        PUSH   TABLE_DIF
         0040  410F00001C         R        AND    TMP0,IN_VAL,#0FH
         0045  C81C                        PUSH   TMP0
         0047  EF0000             E        CALL   DMPY
         004A  0E041C                      SHRAL  TMP0,#4H
         004D  A01E0E             R        LD     OUT_DIF+2H,TMP2
         0050  A01C0C             R        LD     OUT_DIF,TMP0
                                          ;   STATEMENT     19
         0053  A00220             R        LD     TMP4,TABLE_LOW
         0056  0620                        EXT    TMP4
         0058  641C20                      ADD    TMP4,TMP0
         005B  A41E22                      ADDC   TMP6,TMP2
         005E  0E0420                      SHRAL  TMP4,#4H
         0061  A02008             R        LD     OUT,TMP4
                                          ;   STATEMENT     20
         0064  B1FF1C                      LDB    TMP0,#0FFH
         0067  DB02                        BC     @0003
         0069  111C                        CLRB   TMP0
         006B                       @0003:
```

270658-70

```
                006B  981C00                      CMPB  R0,TMP0
                006E  D705                        BNE   @0001
                                                  ;     STATEMENT     21
                0070  A0200A            R         LD    RESULT,TMP4
                0073  2005                        BR    @0002
                                                  ;     STATEMENT     22
                0075                    @0001:
                0075  A0080A            R         LD    RESULT,OUT
                0078  070A              R         INC   RESULT
                                                  ;     STATEMENT     23
                007A                    @0002:
                007A  27AA                        BR    LOOP
                                                  ;     STATEMENT     24
                                                  END


MODULE INFORMATION:

     CODE AREA SIZE                = 005AH     90D
     CONSTANT AREA SIZE            = 0022H     34D
     DATA AREA SIZE                = 0000H      0D
     STATIC REGS AREA SIZE         = 0012H     18D


PL/M-96 COMPILER     PLMEX1   PLM-96 Example Code for Table Lookup
                     ASSEMBLY LISTING OF OBJECT CODE


     OVERLAYABLE REGS AREA SIZE = 0000H       0D
     MAXIMUM STACK SIZE         = 0006H       6D
     48 LINES READ

PL/M-96 COMPILATION COMPLETE.       0 WARNINGS,     0 ERRORS
```

270658-71

```
MCS-96 MACRO ASSEMBLER    MULT.APT:  16*16 multiply procedure for PLM-96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:MULT.A96
OBJECT FILE:  :F3:MULT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                   1   $TITLE('MULT.APT:  16*16 multiply procedure for PLM-96')
                                   2
                                   3
     0018                          4           SP      EQU     18H:word
                                   5
     0000                          6   rseg
                                   7           EXTRN   PLMREG  :long
                                   8
     0000                          9   cseg
                                  10
                                  11           PUBLIC  DMPY        ; Multiply two integers and return a
                                  12                               ; longint result in AX, DX registers
                                  13
     0000 CC04            E      14   DMPY:   POP     PLMREG+4            ; Load return address
     0002 CC00            E      15           POP     PLMREG              ; Load one operand
     0004 FE6E1900        E      16           MUL     PLMREG,[SP]+    ; Load second operand and increment SP
                                  17
     0008 E304            E      18           BR      [PLMREG+4]      ; Return to PLM code.
     000A                        19   END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270658-72

```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES:  :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE:  :F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   ROM(2080H-3FFFH)


INPUT MODULES INCLUDED:
   :F3:PLMEX1.OBJ(PLMEX)   12/25/84
   :F3:MULT.OBJ(MULT)   12/25/84
   PLM96.LIB(PLMREG)   11/02/83


SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

             TYPE     BASE      LENGTH    ALIGNMENT     MODULE NAME
             ----     ----      ------    ---------     -----------

**RESERVED*           0000H     001AH
*** GAP ***           001AH     0002H
             REG      001CH     0008H     ABSOLUTE      PLMREG
             REG      0024H     0012H     WORD          PLMEX
             STACK    0036H     0006H     WORD
*** GAP ***           003CH     2044H
             CODE     2080H     0003H     ABSOLUTE      PLMEX
*** GAP ***           2083H     0001H
             CODE     2084H     007CH     WORD          PLMEX
             CODE     2100H     000AH     BYTE          MULT
*** GAP ***           210AH     DEF6H
```

270658-73

```
SYMBOL TABLE FOR :F3:PLMOUT.OBJ(PLMEX).

ATTRIBUTES              VALUE     NAME
----------              -----     ----

                                  PUBLICS:
REG     WORD            0024H       IN_VAL
REG     INTEGER         0026H       TABLE_LOW
REG     INTEGER         0028H       TABLE_HIGH
REG     INTEGER         002AH       TABLE_DIF
REG     INTEGER         002CH       OUT
REG     INTEGER         002EH       RESULT
REG     LONGINT         0030H       OUT_DIF
REG     WORD            0034H       TEMP
CODE    ENTRY           2100H       DMPY
REG     LONG            001CH       PLMREG
NULL    NULL            003CH       MEMORY
NULL    NULL            1FC4H       ?MEMORY_SIZE

                                  MODULE: PLMEX

                                  MODULE: MULT

                                  MODULE: PLMREG

RL96 COMPLETED,   0 WARNING(S),   0 ERROR(S)
```

270658-74

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  F3:PULSE.A96
OBJECT FILE:  F3:PULSE.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                LINE      SOURCE STATEMENT
                                  1   $TITLE('PULSE.A96: Measuring pulses using the HSI unit')
                                  2
                                  3   $INCLUDE(DEMO96.INC)
                              =1  4   $nolist  ;  Turn listing off for include file
                              =1  52          ;  End of include file
                                  53
    0028                          54  rseg    at  28H
                                  55
    0028                          56          HIGH_TIME:      dsw     1
    002A                          57          LOW_TIME:       dsw     1
    002C                          58          PERIOD:         dsw     1
    002E                          59          HI_EDGE:        dsw     1
    0030                          60          LO_EDGE:        dsw     1
                                  61
                                  62
                                  63
    2080                          64  cseg    at  2080H
                                  65
                                  66
    2080 A1000118                 67          LD      SP, #100H
    2084 B10115                   68          LDB     IOCO, #00000001B        ; Enable HSI 0
    2087 B10F03                   69          LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge
                                  70
    208A 442A282C                 71  wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME
    208E 3E1603                   72          JBS     IOS1, 6, contin         ; If FIFO is full
    2091 3716F6                   73          JBC     IOS1, 7, wait   ; Wait while no pulse is entered
                                  74
    2094 B0061C                   75  contin: LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                  76                                          ;   HSI_TIME clears HSI_STATUS
                                  77
    2097 A00420                   78          LD      BX, HSI_TIME            ; Load the HSI_TIME
                                  79
    209A 391C09                   80          JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high
                                  81
    209D C03020                   82  hsi_lo: ST      BX, LO_EDGE
    20A0 482E3028                 83          SUB     HIGH_TIME, LO_EDGE, HI_EDGE
    20A4 27E4                     84          BR      wait
                                  85
                                  86
    20A6 C02E20                   87  hsi_hi: ST      BX, HI_EDGE
```

270658-75

```
20A9 48302E2A                88          SUB     LOW_TIME, HI_EDGE, LO_EDGE
20AD 27DB                    89          BR      wait
                             90
20AF                         91          END
ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270658-76

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3 ENHSI.A96
OBJECT FILE:   F3 ENHSI.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE ('ENHSI A96  ENHANCED HSI PULSE ROUTINE')
                                    2
                                    3   $INCLUDE(DEMO96.INC)
                                =1  4   $nolist  ;  Turn listing off for include file
                                =1  52          ;  End of include file
                                    53
    0028                            54  RSEG AT 28H
                                    55
    0028                            56          TIME:           DSW 1
    002A                            57          LAST_RISE:      DSW 1
    002C                            58          LAST_FALL:      DSW 1
    002E                            59          HSI_SO:         DSB 1
    002F                            60          IOS1_BAK:       DSB 1
    0030                            61          PERIOD:         DSW 1
    0032                            62          LOW_TIME:       DSW 1
    0034                            63          HIGH_TIME:      DSW 1
    0036                            64          COUNT:          DSW 1
                                    65
    2080                            66  cseg    at      2080H
                                    67
    2080 A1000118                   68  init:   LD      SP,#100H
                                    69
    2084 B12516                     70          LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                    71                                  ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
                                    72
    2087 B19903                     73          LDB     HSI_MODE,#10011001B     ; set hsi.1 -; hsi.0 +
    208A B10715                     74          LDB     IOCO,#00000111B         ; Enable  hsi 0,1
                                    75                                          ; T2 CLOCK=T2CLK,  T2RST=T2RST
                                    76                                          ; Clear timer2
                                    77
                                    78
    208D 717F2F                     79  wait:   ANDB    IOS1_BAK,#01111111B     ; Clear IOS1_BAK.7
    2090 90162F                     80          ORB     IOS1_BAK,IOS1           ; Store into temp to avoid clearing
                                    81                                          ; other flags which may be needed
    2093 372FF7                     82          JBC     IOS1_BAK,7,wait         ; If hsi is not triggered then
                                    83                                          ; jump to wait
                                    84
    2096 5155062E                   85          ANDB    HSI_SO,HSI_STATUS,#01010101B
    209A A00428                     86          LD      TIME, HSI_TIME
                                    87
```

270658-77

```
209D 382E05                    88          JBS     HSI_SO,0,a_rise
20A0 3A2E0F                    89          JBS     HSI_SO,2,a_fall
20A3 201A                      90          BR      no_cnt
                               91
20A5 482C2832                  92  a_rise: SUB     LOW_TIME, TIME, LAST_FALL
20A9 482A2830                  93          SUB     PERIOD, TIME,LAST_RISE
20AD A0282A                    94          LD      LAST_RISE, TIME
20B0 200B                      95          BR      increment
                               96
20B2 482A2834                  97  a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
20B6 482C2830                  98          SUB     PERIOD, TIME,LAST_FALL
20BA A0282C                    99          LD      LAST_FALL, TIME
                              100
20BD                          101  increment:
20BD 0736                     102          INC     COUNT
20BF 27CC                     103  no_cnt: BR      wait
                              104
20C1                          105          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-78

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:HSODRV.A96
OBJECT FILE:  :F3:HSODRV.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE('HSODRV.A96:  Driver module for HSO PWM program')
                                    2
                                    3   HSODRV          MODULE  MAIN, STACKSIZE(8)
                                    4
                                    5
                                    6           PUBLIC  HSO_ON_0 , HSO_OFF_0
                                    7           PUBLIC  HSO_ON_1 , HSO_OFF_1
                                    8           PUBLIC  HSO_TIME , HSO_COMMAND
                                    9           PUBLIC  SP , TIMER1 , IOSO
                                   10
                                   11   $INCLUDE(DEMO96.INC)
                              =1   12   $nolist  ;  Turn listing off for include file
                              =1   60            ;  End of include file
                                   61
     0028                          62   rseg at 28H
                                   63
                                   64           EXTRN   OLD_STAT        :byte
                                   65
     0028                          66           HSO_ON_0:       dsw     1
     002A                          67           HSO_OFF_0:      dsw     1
     002C                          68           HSO_ON_1:       dsw     1
     002E                          69           HSO_OFF_1:      dsw     1
     0030                          70           count:          dsb     1
                                   71
     2080                          72   cseg  at 2080H
                                   73
                                   74           EXTRN   wait    :entry
                                   75
     2080 FA                       76   strt:   DI
     2081 A1000118                 77           LD      SP, #100H
     2085 510F1500           E     78           ANDB    OLD_STAT, IOSO, #0FH
     2089 950F00             E     79           XORB    OLD_STAT, #0FH
                                   80
     208C                          81   initial:
     208C A1000122                 82           LD      CX, #0100H
                                   83
     2090 A100101C                 84   loop:   LD      AX, #1000H
     2094 48221C20                 85           SUB     BX, AX, CX
     2098 A0221C                   86           LD      AX, CX
                                   87
```

270658-79

```
209B C0281C                88          ST      AX, HSO_ON_0
209E C02A20                89          ST      BX, HSO_OFF_0
                           90
20A1 08011C                91          SHR     AX,#1
20A4 080120                92          SHR     BX,#1
20A7 C02C1C                93          ST      AX, HSO_ON_1
20AA C02E20                94          ST      BX, HSO_OFF_1
                           95
20AD EF0000          E     96          CALL    wait
                           97
20B0 0722                  98          INC     CX
20B2 89000F22              99          CMP     CX, #00F00H
20B6 D7D8                 100          BNE     loop
                          101
20B8 27D2                 102          BR      initial
                          103
20BA                      104          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-80

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSOMOD.A96
OBJECT FILE: :F3:HSOMOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT          LINE       SOURCE STATEMENT
                           1   $TITLE('HSOMOD.A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
                           2   $PAGEWIDTH(130)
                           3
                           4   ; This program will provide 3 PWM outputs on HSO pins 0-2
                           5   ; The input parameters passed to the program are:
                           6   ;
                           7   ;            HSO_ON_N    HSO on time for pin N
                           8   ;            HSO_OFF_N   HSO off time for pin N
                           9
                          10   ;        Where:  Times are in timer1 cycles
                          11   ;                N takes values from 0 to 3
                          12
                          13   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          14
                          15
                          16   ;; NOTE: Use this file to replace the declaration section of
                          17   ;           the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
                          18   ;           the line prior to the label "wait".  Also change the last
                          19   ;           branch in the program to a "RET".
                          20
    0000                  21   RSEG
                          22
    0000                  23           D_STAT:         DSB     1
                          24           extrn   HSO_ON_0 :word , HSO_OFF_0 :word
                          25           extrn   HSO_ON_1 :word , HSO_OFF_1 :word
                          26           extrn   HSO_TIME :word , HSO_COMMAND :byte
                          27           extrn   TIMER1   :word , IOSO       :byte
                          28           extrn   SP       :word
                          29
                          30           public  OLD_STAT
    0001                  31           OLD_STAT:       dsb     1
    0002                  32           NEW_STAT:       dsb     1
                          33
                          34
    0000                  35   cseg
                          36           PUBLIC  wait
                          37
    0000 3E00FD      E    38   wait:   JBS     IOSO, 6, wait           ; Loop until HSO holding register
    0003 FD               39           NOP                             ; is empty
                          40
                          41           ; For opperation with interrupts 'store_stat:' would be the
                          42           ; entry point of the routine.
                          43           ; Note that a DI or PUSHF might have to be added.
                          44
```

270658–81

```
0004                            45  store_stat:
0004 510F0002           E       46          ANDB NEW_STAT, IOS0, #0FH       ; Store new status of HSO
0008 980201             R       47          CMPB OLD_STAT, NEW_STAT
000B DFF3                       48          JE   wait
000D 940201             R       49          XORB OLD_STAT, NEW_STAT
                                50
                                51
0010                            52  check_0:
0010 300113             R       53          JBC  OLD_STAT, 0, check_1       ; Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209             R       54          JBS  NEW_STAT, 0, set_off_0
                                55
0016                            56  set_on_0:
0016 B13000             E       57          LDB  HSO_COMMAND, #00110000B    ; Set HSO for timer1, set pin 0
0019 44000000           E       58          ADD  HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
001D 2007                       59          BR   check_1                    ;  + Time for pin to be low
                                60
001F                            61  set_off_0:
001F B11000             E       62          LDB  HSO_COMMAND, #00010000B    ; Set HSO for timer1, clear pin 0
0022 44000000           E       63          ADD  HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
                                64                                          ;  + Time for pin to be high
0026                            65  check_1:
0026 310113             R       66          JBC  OLD_STAT, 1, check_done    ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209             R       67          JBS  NEW_STAT, 1, set_off_1
                                68
002C                            69  set_on_1:
002C B13100             E       70          LDB  HSO_COMMAND, #00110001B    ; Set HSO for timer1, set pin 1
002F 44000000           E       71          ADD  HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
0033 2007                       72          BR   check_done
                                73
0035                            74  set_off_1:
0035 B11100             E       75          LDB  HSO_COMMAND, #00010001B    ; Set HSO for timer1, clear pin 1
0038 44000000           E       76          ADD  HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
                                77                                          ;  + Time for pin to be high
003C                            78  check_done:
003C B00201             R       79          LDB  OLD_STAT, NEW_STAT         ; Store current status and
                                80                                          ; wait for interrupt flag
                                81
003F F0                         82          RET
                                83              use "BR  wait" if this routine is used with the driver
                                84
0040                            85          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-82

A.6. PWM Using the HSO (Continued)

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:SP.A96
OBJECT FILE:   F3:SP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1
                                    2   $TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')
                                    3
                                    4   $INCLUDE(DEMO96.INC)
                               =1   5   $nolist  ;  Turn listing off for include file
                               =1  53            ;  End of include file
                                   54
    0028                           55   rseg     at 28H
                                   56
    0028                           57            CHR:      dsb     1
    0029                           58            SPTEMP:   dsb     1
    002A                           59            TEMP0:    dsb     1
    002B                           60            TEMP1:    dsb     1
    002C                           61            RCV_FLAG:         dsb     1
                                   62
    200C                           63   cseg     at 200CH
                                   64
    200C 9C20                      65            DCW       ser_port_int
                                   66
    2080                           67   cseg     at 2080H
                                   68
    2080 A1000118                  69            LD        SP, #100H
                                   70
    2084 B12016                    71            LDB       IOC1, #00100000B                ; Set P2.0 to TXD
                                   72
                                   73            ; Baud rate = input frequency / (64*baud_val)
                                   74            ; baud_val  = (input frequency/64) / baud rate
                                   75
                                   76
      0027                         77   baud_val          equ     39              ; 39 = (12,000,000/64)/4800 baud
                                   78
      0080                         79   BAUD_HIGH         equ     ((baud_val-1)/256) OR 80H       ; Set MSB to 1
      0026                         80   BAUD_LOW          equ     (baud_val-1) MOD 256
                                   81
                                   82
    2087 B1260E                    83            LDB       BAUD_REG, #BAUD_LOW
    208A B1800E                    84            LDB       BAUD_REG, #BAUD_HIGH
                                   85
```

270658-83

```
208D B14911              86              LDB     SPCON, #01001001B       ; Enable receiver, Mode 1
                         87
                         88                      ; The serial port is now initialized
                         89
                         90
2090 C42807              91              STB     SBUF, CHR               ; Clear serial Port
2093 B1202A              92              LDB     TEMPO, #00100000B       ; Set TI-temp
                         93
2096 B14008              94              LDB     INT_MASK, #01000000B    ; Enable Serial Port Interrupt
2099 FB                  95              EI
209A 27FE                96      loop:   BR      loop                    ; Wait for serial port interrupt
                         97
                         98
209C                     99      ser_port_int:
209C F2                 100              PUSHF
209D                    101      rd_again:                               ; This section of code can be replaced
209D B01129             102              LDB     SPTEMP, SPSTAT          ; with "ORB  TEMPO, SP_STAT" when the
20A0 90292A             103              ORB     TEMPO, SPTEMP           ; serial port TI and RI bugs are fixed
20A3 716029             104              ANDB    SPTEMP,#01100000B
20A6 D7F5               105              JNE     rd_again        ; Repeat until TI and RI are properly cleared
                        106
20A8                    107      get_byte:
20A8 362A09             108              JBC     TEMPO, 6, put_byte      ; If RI-temp is not set
20AB C42807             109              STB     SBUF, CHR               ; Store byte
20AE 71BF2A             110              ANDB    TEMPO, #10111111B       ; CLR RI-temp
20B1 B1FF2C             111              LDB     RCV_FLAG, #0FFH         ; Set bit-received flag
                        112
20B4                    113      put_byte:
20B4 302C18             114              JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
20B7 352A15             115              JBC     TEMPO, 5, continue      ; If TI was not set
20BA B02807             116              LDB     SBUF, CHR               ; Send byte
20BD 71DF2A             117              ANDB    TEMPO, #11011111B       ; CLR TI-temp
                        118
20C0 717F28             119              ANDB    CHR, #01111111B         ; This section of code appends
20C3 990D28             120              CMPB    CHR,#0DH                ; an LF after a CR is sent
20C6 D705               121              JNE     clr_rcv
20C8 B10A28             122              LDB     CHR, #0AH
20CB 2002               123              BR      continue
                        124
20CD                    125      clr_rcv:
20CD 112C               126              CLRB    RCV_FLAG                ; Clear bit-received flag
                        127
20CF                    128      continue:
20CF F3                 129              POPF
20D0 F0                 130              RET
                        131
20D1                    132              END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270658-84

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:ATOD.A96
OBJECT FILE:  :F3:ATOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE('ATOD.A96:  SCANNING THE A TO D CHANNELS')
                                    2
                                    3   $INCLUDE(DEMO96.INC)
                                =1  4   $nolist  ;  Turn listing off for include file
                                =1  52          ;  End of include file
                                    53
    0028                            54  RSEG    at  28H
                                    55
      0020                          56          BL      EQU     BX:BYTE
      001E                          57          DL      EQU     DX:BYTE
                                    58
    0028                            59  RESULT_TABLE:
    0028                            60          RESULT_1:       dsw     1
    002A                            61          RESULT_2:       dsw     1
    002C                            62          RESULT_3:       dsw     1
    002E                            63          RESULT_4:       dsw     1
                                    64
                                    65
    2080                            66  cseg    at  2080H
                                    67
                                    68
    2080 A1000118                   69  start:  LD      SP, #100H       ; Set Stack Pointer
    2084 0120                       70          CLR     BX
                                    71
    2086 55082002                   72  next:   ADDB    AD_COMMAND,BL, #1000B       ; Start conversion on channel
                                    73                                              ; indicated by BL register
                                    74
    208A FD                         75          NOP             ; Wait for conversion to start
    208B FD                         76          NOP
    208C 3B02FD                     77  check:  JBS     AD_RESULT_LO, 3, check  ; Wait while A to D is busy
                                    78
    208F B0021C                     79          LDB     AL, AD_RESULT_LO        ; Load low order result
    2092 B0031D                     80          LDB     AH, AD_RESULT_HI        ; Load high order result
                                    81
    2095 5420201E                   82          ADDB    DL, BL, BL              ; DL=BL*2
    2099 AC1E1E                     83          LDBZE   DX, DL
    209C C31E281C                   84          ST      AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2
                                    85
    20A0 1720                       86          INCB    BL              ; Increment BL modulo 4
```

270658-85

```
20A2 710320                     87              ANDB    BL, #03H
                                88
20A5 27DF                       89              BR      next
                                90
20A7                            91              END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270658-86

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:A2DHSO.A96
OBJECT FILE: :F3:A2DHSO.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                                1   $TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')
                                2
                                3   ; This program will provide 3 PWM outputs on HSO pins 0-2
                                4   ; and one on the PWM.
                                5   ;
                                6   ; The PWM values are determined by the input to the A/D converter.
                                7   ;
                                8   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                9
                               10   $INCLUDE(DEMO96.INC)
                          =1   11   $nolist   ;   Turn listing off for include file
                          =1   59             ;   End of include file
                               60
    0028                       61   RSEG AT 28H
                               62
      001E                     63             DL          EQU         DX:BYTE
                               64
    0028                       65   ON_TIME:
    0028                       66             PWM_TIME_1:             DSW         1
    002A                       67             HSO_ON_0:               DSW         1
    002C                       68             HSO_ON_1:               DSW         1
    002E                       69             HSO_ON_2:               DSW         1
                               70
    0030                       71   RESULT_TABLE:
    0030                       72             RESULT_0:               DSW         1
    0032                       73             RESULT_1:               DSW         1
    0034                       74             RESULT_2:               DSW         1
    0036                       75             RESULT_3:               DSW         1
                               76
    0038                       77             NXT_ON_T:               DSW         1
    003A                       78             NXT_OFF_0:              DSW         1
    003C                       79             NXT_OFF_1:              DSW         1
    003E                       80             NXT_OFF_2:              DSW         1
    0040                       81             COUNT:                  DSL         1
    0044                       82             AD_NUM:                 DSW         1      ; Channel being converted
    0046                       83             TMP:                    DSW         1
    0048                       84             HSO_PER:                DSW         1
    004A                       85             LAST_LOAD:              DSB         1
                               86
```

270658-87

```
2000                          87  cseg    AT 2000H
                              88
2000 8020                     89          DCW     start           ; Timer_ovf_int
2002 1D21                     90          DCW     Atod_done_int
2004 8020                     91          DCW     start           ; HSI_data_int
2006 CC20                     92          DCW     HSO_exec_int
                              93
                              94  $EJECT
                              95
2080                          96  cseg    AT 2080H
                              97
2080 A1000118                 98  start:  LD      SP, #100H       ; Set Stack Pointer
2084 011C                     99          CLR     AX
2086 051C                    100  wait:   DEC     AX              ; wait approx. 0.2 seconds for
2088 D7FC                    101          JNE     wait            ; SBE to finish communications
                             102
208A 1144                    103          CLRB    AD_NUM
                             104
208C A1800028                105          LD      PWM_TIME_1, #080H
2090 A1000148                106          LD      HSO_PER, #100H
2094 A140002A                107          LD      HSO_ON_0, #040H
2098 A180002C                108          LD      HSO_ON_1, #080H
209C A1C0002E                109          LD      HSO_ON_2, #0C0H
                             110
20A0 4500010A38              111          ADD     NXT_ON_T,  Timer1, #100H
                             112
20A5 B13606                  113          LDB     HSO_COMMAND, #00110110B     ; Set HSO for timer1, set pin 0,1
20A8 A03804                  114          LD      HSO_TIME, NXT_ON_T          ; with interrupt
20AB FD                      115          NOP
20AC FD                      116          NOP
20AD B12206                  117          LDB     HSO_COMMAND, #00100010B     ; Set HSO for timer1, set pin 2
20B0 643804                  118          ADD     HSO_TIME, NXT_ON_T          ; without interrupt
                             119
20B3 91074A                  120          ORB     LAST_LOAD, #00000111B   ; Last loaded value was set all pins
20B6 B10A08                  121          LDB     INT_MASK, #00001010B    ; Enable HSO and A/D interrupts
20B9 B10A09                  122          LDB     INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
20BC FB                      123          EI
                             124
20BD 91010F                  125  loop:   ORB     Port1, #00000001B       ; set P1.0
20C0 65010040                126          ADD     COUNT, #01
20C4 A40042                  127          ADDC    COUNT+2, zero
20C7 71FE0F                  128          ANDB    Port1, #11111110B       ; clear P1.0
20CA 27F1                    129          BR      loop
                             130
                             131  $EJECT
```

270658-88

```
                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                   ;;;;;;;;;;;;;;;;;        HSO  EXECUTED  INTERRUPT        ;;;;;;;;;;;;;;;;;;;;;;
                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

20CC               HSO_exec_int:
20CC F2                     PUSHF
20CD 91020F                 ORB     Port1, #00000010B          ; Set p1.1

20D0 48380A46               SUB     TMP,TIMER1, NXT_ON_T
20D4 880046                 CMP     TMP,ZERO
20D7 DE19                   JLT     set_off_times

20D9               set_on_times:
20D9 644838                 ADD     NXT_ON_T, HSO_PER
20DC B13606                 LDB     HSO_COMMAND, #00110110B    ; Set HSO for timer1, set pin 0,1
20DF A03804                 LD      HSO_TIME, NXT_ON_T
20E2 FD                     NOP
20E3 FD                     NOP
20E4 B12206                 LDB     HSO_COMMAND, #00100010B    ; Set HSO for timer1, set pin 2
20E7 A03804                 LD      HSO_TIME, NXT_ON_T

20EA 91074A                 ORB     LAST_LOAD, #00000111B      ; Last loaded value was all ones

20ED B02817                 LDB     PWM_CONTROL, PWM_TIME_1     ; Now is as good a time as any
                                                                ; to update the PWM reg
20F0 2026                   BR      check_done


20F2               set_off_times:
20F2 304A23                 JBC     LAST_LOAD, 0, check_done

20F5 442A383A               ADD     NXT_OFF_0, NXT_ON_T, HSO_ON_0
20F9 B11006                 LDB     HSO_COMMAND, #00010000B     ; Set HSO for timer1, clear pin 0
20FC A03A04                 LD      HSO_TIME, NXT_OFF_0

20FF FD                     NOP
2100 442C383C               ADD     NXT_OFF_1, NXT_ON_T, HSO_ON_1
2104 B11106                 LDB     HSO_COMMAND, #00010001B     ; Set HSO for timer1, clear pin 1
2107 A03C04                 LD      HSO_TIME, NXT_OFF_1

210A FD                     NOP
210B 442E383E               ADD     NXT_OFF_2, NXT_ON_T, HSO_ON_2
210F B11206                 LDB     HSO_COMMAND, #00010010B      ; Set HSO for timer1, clear pin 2
2112 A03E04                 LD      HSO_TIME, NXT_OFF_2

2115 71F84A                 ANDB    LAST_LOAD, #11111000B   ; Last loaded value was all 0s

2118               check_done:
2118 71FD0F                 ANDB    Port1, #11111101B          ; Clear P1.1
```

270658-89

```
    211B F3                      POPF
    211C F0                      RET

                         $EJECT

                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;;;;;;;;;;;;;;;;;       A TO D  COMPLETE  INTERRUPT         ;;;;;;;;;;;;;;;;;;;;;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    211D                 ATOD_done_int:
    211D F2                      PUSHF
    211E 91040F                  ORB     Port1, #00000100B        ; Set P1.2

    2121 51C0021C                ANDB    AL, AD_RESULT_LO,#11000000B      ; Load low order result
    2125 B0031D                  LDB     AH, AD_RESULT_HI                 ; Load high order result
    2128 5444441E                ADDB    DL, AD_NUM, AD_NUM               ; DL= AD_NUM *2
    212C AC1E1E                  LDBZE   DX, DL
    212F C31E301C                ST      AX, RESULT_TABLE[DX]     ; Store result indexed by DX

    2133 99401C                  CMPB    AL, #01000000B
    2136 D107                    JNH     no_rnd                   ; Round up if needed
    2138 99FF1D                  CMPB    AH,#0FFH         ; Don't increment if AH=0FFH
    213B DF02                    JE      no_rnd
    213D 171D                    INCB    AH

    213F B01D1C          no_rnd: LDB     AL, AH           ; Align byte and change to word
    2142 111D                    CLRB    AH
    2144 C31E281C                ST      AX, ON_TIME[DX]

    2148 1744                    INCB    AD_NUM
    214A 710344                  ANDB    AD_NUM, #03H                     ; Keep AD_NUM between 0 and 3

    214D 55084402        next:   ADDB    AD_COMMAND, AD_NUM, #1000B       ; Start conversion on channel
                                                                          ; indicated by AD_NUM register
    2151 71FB0F                  ANDB    Port1, #11111011B        ; Clear P1.2
    2154 F3                      POPF
    2155 F0                      RET


    2156                         END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

270658-90

APPENDIX C
SOFTWARE SERIAL PORT

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:SWPORT.A96
OBJECT FILE:  :F3:SWPORT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')
                                    2
                                    3   ; This module provides a software implemented asynchronous serial port
                                    4   ; for the 8096.  HSO.5 is used for transmit data.  HSI.2 is used for
                                    5   ; receive data.  Note: the choice of HSO.5 and HSI.2 is arbitrary).
                                    6
                                    7   $INCLUDE(DEMO96.INC)
                                =1  8   $nolist  ;  Turn listing off for include file
                                =1 56             ;  End of include file
                                   57
                                   58   ;        VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
                                   59   ;        ===========================================
                                   60   ;
0000                               61           rseg
                                   62
0000                               63   ios1_save:       dsb 1     ; Used to save contents of ios1
0001                               64   rcve_state:      dsb 1
   0001                            65     rxrdy          equ 1     ; indicates receive done
   0002                            66     rxoverrun      equ 2     ; indicates receive overflow
   0004                            67     rip            equ 4     ; receive in progress flag
0002                               68   rcve_buf:        dsb 1     ; used to double buffer receive data
0003                               69   rcve_reg:        dsb 1     ; used to deserialize receive
0004                               70   sample_time:     dsw 1     ; records last receive sample time
                                   71
0006                               72   serial_out:      dsw 1     ; Holds the output character+framing (start and
                                   73                             ;   stop bits) for transmit process.
   0008                            74   baud_count:      dsw 1     ; Holds the period of one bit in units
                                   75                             ;   of T1 ticks.
   000A                            76   txd_time:        dsw 1     ; Transition time of last Txd bit that was
                                   77                             ;   sent to the CAM.
   000C                            78   char:            dsb 1     ; for test only
                                   79   ;
                                   80   ;        COMMANDS ISSUED TO THE HSO UNIT
                                   81   ;        ==============================
                                   82   ;
   0035                            83   mark_command     equ     0110101b        ; timer1,set,interrupt on 5
   0015                            84   space_command    equ     0010101b        ; timer1,clr,interrupt on 5
   0018                            85   sample_command   equ     0011000b        ; software timer 0
                                   86
                                   87   $eject
```

270658-91

```
2080                        88          cseg at 2080h
                            89  ;
2080                        90  reset_loc:
                            91  ; The 8096 starts executing here on reset, the program will initialize the
                            92  ; the software serial port and run a simple test to excercize it.
                            93  ;
2080 FA                     94          di
2081 A1F00018               95          ld      sp,#0f0h
2085 C9C012                 96          push    #4800
2088 EF0000          R      97          call    setup_serial_port
208B B16C08                 98          ldb     int_mask,#01101100b     ; serial, swt,hso,hsi
208E FB                     99          ei
                           100
                           101  ;
208F                       102  test1:
                           103  ; A simple test of the serial port routines.
                           104  ; While no characters are received an incrementing pattern is sent to the
                           105  ; serial output.  When a character is received the incrementing pattern
                           106  ; "jumps" to the character receved and proceeds from there.
                           107  ;
  000D                     108          CR      equ     0DH             ; Carriage return
208F B10D0C          R     109          ldb     char,#CR
2092                       110  testiloop:
2092 AC0C1C          R     111          ldbze   ax,char
2095 C81C                  112          push    ax
2097 EF3000          R     113          call    char_out
                           114
209A 990D0C          R     115          cmpb    char,#CR                ; Pause on Carriage return
209D D706                  116          bne     nopause
209F 011C                  117          clr     ax
20A1                       118  pause:
20A1 071C                  119          inc     ax
20A3 D7FC                  120          bne     pause
20A5                       121  nopause:
                           122
20A5 170C            R     123          incb    char
20A7                       124  test2:
20A7 EF4400          R     125          call    csts                    ; char ready?
20AA 98001C                126          cmpb    al,0
20AD DFE3                  127          be      testiloop               ;   loop if not
20AF EF4C00          R     128          call    char_in
20B2 B01C0C          R     129          ldb     char,al
20B5 27DB                  130          br      testiloop
                           131  $eject
```

270658-92

```
                                132
0000                            133             cseg
                                134     ;
0000                            135     setup_serial_port:
                                136     ; Called on system reset to intiate the software serial port.
                                137     ;
0000 CC22                       138             pop     cx                      ; the return address
0002 CC20                       139             pop     bx                      ; the baud rate (in decimal)
0004 A107001E                   140             ld      dx,#0007h               ; dx:ax =500,000 (assumes 12 Mhz crystal)
0008 A120A11C                   141             ld      ax,#0A120h
000C BC201C                     142             divu    ax,bx                   ; calculate the baud count (500,000/baudrate)
000F C0081C                  R  143             st      ax,baud_count
0012 C00600                  R  144             st      0,serial_out            ; clear serial out
0015 B16016                     145             ldb     ioc1,#01100000b         ; Enable HSO.5 and Txd
0018 3E15FD                     146             bbs     ios0,6,$                ; Wait for room in the HSO CAM
                                147                                             ; and issue a MARK command.
001B 44140A0A                R  148             add     txd_time,timer1,20
001F B13506                     149             ldb     hso_command,#mark_command
0022 A00A04                  R  150             ld      hso_time,txd_time
0025 1102                    R  151             clrb    rcve_buf                ; clear out the receive variables
0027 1103                    R  152             clrb    rcve_reg
0029 1101                    R  153             clrb    rcve_state
002B EF4800                     154             call    init_receive            ; setup to detect a start bit
002E E322                       155             br      [cx]                    ; return
                                156     $eject
                                157     ;
0030                            158     char_out:
                                159     ; Output character to the software serial port
                                160     ;
0030 CC22                       161             pop     cx                      ; the return address
0032 CC20                       162             pop     bx                      ; the character for output
0034 B10121                     163             ldb     (bx+1),#01h             ; add the start and stop bits
0037 642020                     164             add     bx,bx                   ;   to the char and leave as 16 bit
003A                            165     wait_for_xmit:
003A 880006                  R  166             cmp     serial_out,0            ; wait for serial_out=0 (it will be cleared by
003D D7FB                       167             bne     wait_for_xmit           ;   the hso interrupt process)
003F C00620                  R  168             st      bx,serial_out           ; put the formatted character in serial_out
0042 E322                       169             br      [cx]                    ; return to caller
                                170     ;
0044                            171     csts:
                                172     ; Returns "true" (ax<>0) if char_in has a character.
                                173     ;
0044 011C                       174             clr     ax
0046 300102                  R  175             bbc     rcve_state,0,csts_exit
0049 071C                       176             inc     ax
004B                            177     csts_exit:
004B F0                         178             ret
                                179     ;
004C                            180     char_in:
```

270658-93

```
                        181   ; Get a character from the software serial port
                        182   ;
                        183                                   ; wait for character ready
004C 3001FD          R  184           bbc     rcve_state,0,char_in
004F F2                 185           pushf                   ; set up a critical region
0050 71FE01          R  186           andb    rcve_state,#not(rxrdy)
0053 AC021C          R  187           ldbze   al,rcve_buf
0056 F3                 188           popf                    ; leave the critical region
0057 F0                 189           ret
                        190   $eject
                        191   ;
0058                    192   hso_isr:
                        193   ; Fields the hso interrupts and performs the serialization of the data.
                        194   ; Note:  this routine would be incorporated into the hso service strategy
                        195   ;        for an actual system.
                        196
2006                    197           cseg    at 2006h
2006 5800            R  198           dcw     hso_isr         ; Set up vector
                        199
0058                    200           cseg
0058 F2                 201           pushf
0059 64080A          R  202           add     txd_time,baud_count
005C 880006          R  203           cmp     serial_out,0    ; if character is done send a mark
005F DF0D               204           be      send_mark
0061 080106          R  205           shr     serial_out,#1   ; else send bit 0 of serial_out and shift
0064 DB08               206           bc      send_mark       ;   serial_out left one place.
0066                    207   send_space:
0066 B11506             208           ldb     hso_command,#space_command
0069 A00A04          R  209           ld      hso_time,txd_time
006C 2006               210           br      hso_isr_exit
006E                    211   send_mark:
006E B13506             212           ldb     hso_command,#mark_command
0071 A00A04          R  213           ld      hso_time,txd_time
                        214
0074                    215   hso_isr_exit:
0074 F3                 216           popf
0075 F0                 217           ret
                        218   $eject
                        219   ;
0076                    220   init_receive:
                        221   ; Called to prepare the serial input process to find the leading edge of
                        222   ; a start bit.
                        223   ;
0076 B10015             224           ldb     ioc0,#00000000b         ; disconnect change detector
0079 B12003             225           ldb     hsi_mode,#00100000b     ; negative edges on HSI.2
007C                    226   flush_fifo:
007C 901600          R  227           orb     ios1_save,ios1
007F 37000B          R  228           bbc     ios1_save,7,flush_fifo_done
0082 B0061C             229           ldb     al,hsi_status
0085 A0041C             230           ld      ax,hsi_time             ; trash the fifo entry
```

270658-94

```
0088 717F00      R     231             andb    ios1_save,#not(80h)     ; clear bit 7.
008B 27EF              232             br      flush_fifo
008D                   233     flush_fifo_done:
008D B11015            234             ldb     ioc0,#00010000b         ; connect HSI.2 to detector
0090 F0                235             ret
                       236
                       237
                       238     ;
0091                   239     hsi_isr:
                       240     ; Fields interrupts from the HSI unit, used to detect the leading edge
                       241     ; of the START bit
                       242     ; Note: this routine would be incorporated into the HSI strategy of an actual
                       243     ; system.
                       244     ;
2004                   245             cseg at 2004h
2004 9100        R     246             dcw     hsi_isr                 ; setup the interrupt vector
                       247
0091                   248             cseg
0091 F2                249             pushf
0092 C81C              250             push    ax
0094 B0061C            251             ldb     al,hsi_status
0097 A00404      R     252             ld      sample_time,hsi_time
009A 341C15            253             bbc     al,4,exit_hsi
009D 3F15FD            254             bbs     ios0,7,$                ; wait for room in HSO holding reg
00A0 A0081C      R     255             ld      ax,baud_count           ; send out sample command in 1/2
00A3 08011C            256             shr     ax,#1                   ; bit time
00A6 641C04      R     257             add     sample_time,ax
00A9 B11806            258             ldb     hso_command,#sample_command
00AC C00404      R     259             st      sample_time,hso_time
00AF B10015            260             ldb     ioc0,#00000000b         ; disconnect hsi.2 from change detector
00B2                   261     exit_hsi:
00B2 CC1C              262             pop     ax
00B4 F3                263             popf
00B5 F0                264             ret
                       265     $eject
                       266     ;
00B6                   267     software_timer_isr:
                       268     ; Fields the software timer interrupt, used to deserialize the incomming data
                       269     ; Note: this routine would be incorporated into the software timer stategy
                       270     ; in an actual system.
                       271     ;
200A                   272             cseg at 200ah
200A B600        R     273             dcw     software_timer_isr      ; setup vector
                       274
00B6                   275             cseg
00B6 F2                276             pushf
00B7 901600      R     277             orb     ios1_save,ios1
00BA 71FE00      R     278             andb    ios1_save,#not(01h)     ; clear bit 0
00BD 51FC0100    R     279             andb    0,rcve_state,#0fch      ; All bits except rxrdy and overrun=0
00C1 D70C              280             bne     process_data
```

```
00C3                    281   process_start_bit:
00C3 350604             282           bbc     hsi_status,5,start_ok
00C6 2FAE               283           call    init_receive
00C8 2032               284           br      software_timer_exit
00CA                    285   start_ok:
00CA 910401         R   286           orb     rcve_state,#rip ; set receive in progress flag
00CD 2021               287           br      schedule_sample
                        288
00CF                    289   process_data:
00CF 3F010E         R   290           bbs     rcve_state,7,check_stopbit
00D2 180103         R   291           shrb    rcve_reg,#1
00D5 350603             292           bbc     hsi_status,5,datazero
00D8 918003         R   293           orb     rcve_reg,#80h   ; set the new data bit
00DB                    294   datazero:
00DB 751001         R   295           addb    rcve_state,#10h ; increment bit count
00DE 2010               296           br      schedule_sample
                        297
00E0                    298   check_stopbit:
00E0 3506FD             299           bbc     hsi_status,5,$  ; DEBUG ONLY
00E3 B00302         R   300           ldb     rcve_buf,rcve_reg
00E6 910101         R   301           orb     rcve_state,#rxrdy
00E9 710301         R   302           andb    rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2F88               303           call    init_receive
00EE 200C               304           br      software_timer_exit
                        305
00F0                    306   schedule_sample:
00F0 3F15FD             307           bbs     ios0,7,$        ; wait for holding reg empty
00F3 B11806             308           ldb     hso_command,#sample_command
00F6 640804         R   309           add     sample_time,baud_count
00F9 C00404         R   310           st      sample_time,hso_time
                        311
00FC                    312   software_timer_exit:
00FC F3                 313           popf
00FD F0                 314           ret
                        315
                        316
00FE                    317           end

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-96

APPENDIX D
MOTOR CONTROL PROGRAM

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:MOTCON.A96
OBJECT FILE:  :F3:MOTCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT          LINE      SOURCE STATEMENT
                            1  $TITLE ('MOTCON.A96:  Motor Control Example Program')
                            2
                            3  ;       USE WITH C-STEP or later parts
                            4
                            5  ;                                    December 20, 1984
                            6
                            7  $INCLUDE(DEMO96.INC)
                  =1        8  $nolist  ;  Turn listing off for include file
                  =1       56           ;  End of include file
                           57
                           58  ;;;;;;;;;;;     Initial Values
                           59
     001E                  60  min_hsi1_t      equ     30  ; min period for PHA edges in mode1 before mode2
                           61
     003C                  62  min_hsi_t       equ     2*min_hsi1_t
                           63                      ; min period for PHA edges in mode0 before mode1
                           64
     0069                  65  max_hsi1_t      equ     3*min_hsi1_t + min_hsi1_t/2
                           66                      ; max period for PHA edges in mode1 before mode0
                           67
                           68
     006E                  69  HSO0_dly_period equ     110     ; delay for HSO timer 0 (timed count of pulses)
                           70                          ; min period for 5 T2 clocks before mode 1
                           71
     00FA                  72  swt1_dly_period equ     250     ; delay for software timer 1
     00FA                  73  swt2_dly_period equ     250     ; delay for software timer 2
     00FF                  74  max_power       equ     0ffh
     00FF                  75  max_brake       equ     0ffh
     0080                  76  maximum_hold    equ     080H
     04B0                  77  brake_pnt       equ     1200
     0064                  78  position_pnt    equ     100
     0010                  79  velocity_pnt    equ     16
                           80
                           81
   0024                    82  RSEG at 024H
                           83
     0024                  84          tmp:        dsl 1
     0028                  85          timer_2:    dsl 1
```

270658-97

```
002C                    86          tmr2_old:       dsl 1
0030                    87          position:       dsl 1
0034                    88          des_pos:        dsl 1
0038                    89          pos_err:        dsl 1
003C                    90          delta_p:        dsl 1
0040                    91          time:           dsl 1
0044                    92          des_time:       dsl 1
0048                    93          time_err:       dsl 1
                        94
                        95  $EJECT
                        96
004C                    97          last_time_err   dsw 1
004E                    98          last_pos_err:   dsw 1
0050                    99          pos_delta:      dsw 1
0052                   100          time_delta:     dsw 1
0054                   101          last_pos:       dsw 1
0056                   102          last1_time:     dsw 1
0058                   103          last2_time:     dsw 1
005A                   104          boost:          dsw 1
005C                   105          tmp1:           dsw 1
005E                   106          out_ptr:        dsw 1
0060                   107          offset:         dsw 1
0062                   108          nxt_pos:        dsw 1
0064                   109          rpwr:           dsw 1
0066                   110          old_t2:         dsw 1
                       111
0068                   112          direct:         dsb 1   ;  1=forward, 0=reverse
0069                   113          pwm_dir:        dsb 1
006A                   114          hsi_s0:         dsb 1
006B                   115          last_stat:      dsb 1
006C                   116          pwm_pwr:        dsb 1
006D                   117          ios1_bak:       dsb 1
006E                   118          TR_COL:         DSB 1   ; COLLECT TRACE IF TR_COL=00
006F                   119          main_dly:       dsb 1
                       120
0070                   121          max_pwr:        dsw 1
0072                   122          max_brk:        dsw 1
0074                   123          max_hold:       dsw 1
0076                   124          vel_pnt:        dsw 1
0078                   125          brk_pnt:        dsw 1
007A                   126          pos_pnt:        dsw 1
007C                   127          HSOO_dly:       dsw 1
007E                   128          swt1_dly:       dsw 1
0080                   129          swt2_dly:       dsw 1
0082                   130          min_hsi:        dsw 1
0084                   131          min_hsi1:       dsw 1
0086                   132          max_hsi1:       dsw 1
                       133
                       134
0100                   135  dseg at 100H
```

270658-98

APPENDIX D
MARGORP COMPACT ROTOM

```
                        136
0100                    137 mode_view:      dsb     1
0102                    138 count_out:      dsw     1
0104                    139 err_view:       dsw     1
                        140
                        141
                        142 $eject
                        143
                        144 ;       PIN#    PORT    FLAG USAGE
                        145
                        146 ;       22      P1.0    mode0 0   mode1 1   mode2 1  or 0
                        147 ;       23      P1.1          0         0         1     1
                        148 ;       24      P1.2    software timer 2 routine enter/leave
                        149 ;       25      P1.3    Main program toggle
                        150 ;       26      P1.4    HSI overflow toggle
                        151 ;       37      P1.5    software timer 0 routine enter/leave
                        152 ;       38      P1.6    hsi_int enter/leave
                        153 ;       39      P1.7    software timer 1 routine enter/leave
                        154 ;       40      P2.6    Input direction (0=reverse, 1=forward)
                        155 ;       45      P2.7    direction 0=rev, 1=fwd
                        156
2000                    157 cseg    at      2000H
2000 0022               158         dcw     timer_ovf_int
2002 1020               159         dcw     atod_done_int
2004 0424               160         dcw     hsi_data_int
2006 8022               161         dcw     hso_exec_int
2008 1020               162         dcw     hsi_0_int
200A 2022               163         dcw     soft_tmr_int
200C 1020               164         dcw     ser_port_int
200E 1020               165         dcw     external_int
                        166
2010                    167 atod_done_int:
2010                    168 hsi_0_int:
2010                    169 ser_port_int:
2010                    170 external_int:
                        171
2080                    172 cseg    at      2080H
                        173
2080 A1F00018           174 init:   ld      sp,#0F0H
2084 B1FF17             175         ldb     pwm_control,#0FFH
                        176
2087 1168               177         clrb    direct
2089 A170175C           178         ld      tmp1,#6000          ; wait about 3 seconds for motor
208D 055C               179 delay:  dec     tmp1            ;   to come to a stop
208F E068FD             180         djnz    direct,$            ; wait 0.512 milliseconds
2092 88005C             181         cmp     tmp1,zero
2095 D2F6               182         jgt     delay
                        183
2097 B1FF0F             184         ldb     port1,#0FFH
209A B1FF10             185         ldb     port2,#0ffH
```

270658-99

```
209D B12516          186           ldb     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                     187                                   ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
                     188
20A0 71FC0F          189           andb    Port1,#11111100B        ; clear P1.0,1 (set mode 0)
20A3 B19903          190           ldb     HSI_mode,#10011001B     ; set hsi.1,3 -; hsi.0,2 +
20A6 B15715          191           ldb     IOC0,#01010111B         ; Enable all hsi
                     192                                           ; T2 CLOCK=T2CLK, T2RST=T2RST
                     193                                           ; Clear timer2
                     194   $eject
                     195
20A9 A00400          196           ld      zero,hsi_time
20AC 0140            197           clr     time
20AE 0142            198           clr     time+2
20B0 0128            199           clr     timer_2
20B2 012A            200           clr     timer_2+2
20B4 0130            201           clr     position
20B6 0132            202           clr     position+2
20B8 0154            203           clr     last_pos
20BA 0134            204           clr     des_pos
20BC 0136            205           clr     des_pos+2
20BE 0144            206           clr     des_time
20C0 0146            207           clr     des_time+2
20C2 A00A56          208           ld      last1_time,Timer1
20C5 4900085658      209           sub     last2_time,last1_time,#800H
20CA 116D            210           clrb    ios1_bak
20CC 1109            211           clrb    int_pending
20CE A1F0015E        212           ld      out_ptr,#1F0H
20D2 A13C0082        213           ld      min_hsi,#min_hsi_t
20D6 A11E0084        214           ld      min_hsi1,#min_hsi1_t
20DA A1690086        215           ld      max_hsi1,#max_hsi1_t
20DE A16E007C        216           ld      HSO0_dly,#HSO0_dly_period
20E2 A1FA007E        217           ld      swt1_dly,#swt1_dly_period
20E6 A1FA0080        218           ld      swt2_dly,#(swt2_dly_period)
20EA A1FF0070        219           ld      max_pwr,#max_power
20EE A1FF0072        220           ld      max_brk,#max_brake
20F2 A1800074        221           ld      max_hold,#maximum_hold
20F6 A1B00478        222           ld      brk_pnt,#brake_pnt
20FA A164007A        223           ld      pos_pnt,#position_pnt
20FE A1100076        224           ld      vel_pnt,#velocity_pnt
2102 A1002962        225           ld      nxt_pos,#pos_table
2106 B0006C          226           ldb     pwm_pwr,zero
2109 B10169          227           ldb     pwm_dir,#01h            ; FORWARD
                     228
210C B12D08          229           ldb     int_mask,#00101101B     ; Enable tmr_ovf, hsi, swt, HSO.interrupts
210F B13006          230           ldb     hso_command,#30H        ; set HSO_0
2112 447C0A04        231           add     hso_time,timer1,HSO0_dly
2116 FD              232           nop
2117 FD              233           NOP
2118 B13906          234           ldb     hso_command,#39H        ; set swt_1
211B 447E0A04        235           add     hso_time,timer1,swt1_dly
```

270658-A0

```
211F FD                 nop
2120 FD                 nop
2121 B13A06             ldb     hso_command,#3AH        ; set swt_2
2124 44B00A04           add     hso_time,timer1,swt2_dly

2128 A00A40             ld      time,TIMER1
212B A00C2C             ld      tmr2_old,timer2
212E FB                 ei

212F E7CE06             br      main_prog

$eject

;*****************************************************************************
;******                    TIMER  INTERRUPT  SERVICE                    ******
;*****************************************************************************

2200                    CSEG AT 2200H

2200                    timer_ovf_int:
2200 F2                 pushf

2201 90166D             orb     ios1_bak,IOS1
2204 356D05             jbc     ios1_bak,5,tmr_int_done
2207 0742               inc     time+2
2209 71DF6D             andb    ios1_bak,#11011111B     ; clear bit 5
220C                    tmr_int_done:
220C F3                 popf
220D F0                 ret             ; End of timer interrupt routine

;*****************************************************************************
;******          SOFTWARE TIMER INTERRUPT SERVICE ROUTINE           ******
;*****************************************************************************

2220                    CSEG AT 2220H

2220                    soft_tmr_int:
2220 F2                 pushf
2221 90166D             orb     ios1_bak,IOS1
2224                    chk_swt0:
2224 306D03             jbc     ios1_bak,0,chk_swt1
2227 71FE6D             andb    ios1_bak,#11111110B     ; Clear bit 0 - end swt0
                        call    swt0_expired
222A                    chk_swt1:
222A 316D06             jbc     ios1_bak,1,chk_swt2
222D 71FD6D             andb    ios1_bak,#11111101B     ; Clear bit 1
```

270658-A1

```
2230 EFCD03               call    swt1_expired
2233                  chk_swt2:
2233 326D06               jbc     ios1_bak,2,chk_swt3
2236 71FB6D               andb    ios1_bak,#11111011B      ; Clear bit 2
2239 EF4401               call    swt2_expired
223C                  chk_swt3:
223C 346D03               jbc     ios1_bak,4,swt_int_done
223F 71F76D               andb    ios1_bak,#11110111B      ; Clear bit 3
                      ;   call    swt3_expired

2242                  swt_int_done:
2242 F3                   popf
2243 F0                   ret     ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

                      $eject

                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                      ;;;;;;        SOFTWARE TIMER ROUTINE 0                  ;;;;;;;;;;;;
                      ;;;;;;         NOW USING HSO.0 TO TRIGGER               ;;;;;;;;;;;;
                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2280                      CSEG AT 2280H

2280                  hso_exec_int:              ; Check mode -  Update position in mode 2

2280 F2                   PUSHF
2281 B13006               ldb     HSO_COMMAND,#30H
2284 447C0A04             add     HSO_TIME,TIMER1,HSO0_dly

2288 91200F               orb     port1,#00100000B         ; set P1.5
228B A00C28               ld      Timer_2,TIMER2
228E 390F18               jbs     Port1,1,in_mode2

2291                  in_mode1:
2291 4866285C             sub     tmp1,Timer_2,old_t2      ; Check count difference in tmp1
2295 8902005C             cmp     tmp1,#2
2299 D94C                 jh      end_swt0
229B                  set_mode0:
229B 300F49               jbc     Port1,0,end_swt0         ; if already in mode 0
229E 71FC0F               andb    Port1,#11111100B         ; Clear P1.0, P1.1 (set mode 0)
22A1 B15515               ldb     IOC0,#01010101B          ; enable all HSI
22A4 B0006B               ldb     last_stat,zero
22A7 203E                 br      end_swt0

22A9                  in_mode2:
22A9 482C283C             sub     delta_p,timer_2,tmr2_old        ; get timer2 count difference
22AD A0282C               ld      tmr2_old,timer_2

22B0 306808               jbc     direct,0,in_rev
```

270658-A2

```
22B3 643C30         in_fwd: add     position,delta_p
22B6 A40032                 addc    position+2,zero
22B9 2006                   br      chk_mode

22BB 683C30         in_rev: sub     position,delta_p
22BE A80032                 subc    position+2,zero

22C1                chk_mode:
22C1 4866285C               sub     tmp1,Timer_2,old_t2        ; Check count difference in tmp1
22C5 8905005C               cmp     tmp1,#5                    ; set mode1 if count is too low
22C9 D21C                   jgt     end_swt0                   ; count <= 5

22CB                set_mode1:
22CB 71FD0F                 andb    Port1,#11111101B           ; Clear P1.1, set P1.0 (set mode 1)
22CE 91010F                 orb     Port1,#00000001B
22D1 B10515                 ldb     IOC0,#00000101B            ; enable HSI 0 and 1
22D4 A00400                 ld      zero, HSI_TIME
22D7 48840A56               sub     last1_time,Timer1,min_hsi1
                                            ; set up so (time-last2_time)>min_hsi1 on next HSI
                    $EJECT

22DB                clr_hsi:
22DB A00400                 ld      ZERO, HSI_TIME
22DE 717F6D                 andb    ios1_bak,#01111111B        ; clear bit 7
22E1 90166D                 orb     ios1_bak,ios1
22E4 3F6DF4                 jbs     ios1_bak,7,clr_hsi         ; If hsi is triggered then clear hsi

22E7                end_swt0:
22E7 A02866                 ld      old_t2,TIMER_2
22EA 71DF0F                 andb    port1,#11011111B           ; clear P1.5
22ED F3                     POPF
22EE F0                     ret



                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    ;;;;;;           SOFTWARE TIMER ROUTINE 2              ;;;;;;;;;;;
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2380                        CSEG AT 2380H

2380                swt2_expired:
2380 F2                     pushf
2381 B13A06                 ldb     hso_command,#3AH           ; set swt_2
2384 44800A04               add     hso_time,timer1,swt2_dly

2388 91040F                 orb     port1,#00000100B           ; set port 1.2
238B 89FF075E               cmp     out_ptr,#7ffH
238F D104                   bnh     pulsing
2391 A1F0015E               ld      out_ptr,#1f0H
```

270658-A3

```
2395                   pulsing:
2395 306E0C                      jbc     tr_col,0,swt2_done

239B C25F32                      st      position+2,[out_ptr]+    ;  position high, position low
239B C25F30                      st      position,[out_ptr]+

239E C25F68                      st      direct,[out_ptr]+
23A1 C25F6C                      st      pwm_pwr,[out_ptr]+

                                                                  ; store 8 bytes externally

23A4                   swt2_done:
23A4 48560A5C                    sub     tmp1,timer1,last1_time
23A8 8900185C                    cmp     tmp1,#1800H
23AC D104                        jnh     swt2_ret          ; keep (Timer1-last1_time)<2000H

23AE 65001056                    add     last1_time,#1000H
23B2                   swt2_ret
23B2 71FB0F                      andb    port1,#11111011B         ; clear port1.2
23B5 F3                          popf
23B6 F0                          ret

                       $EJECT
                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       ;;;;;               HSI DATA AVAILABLE INTERRUPT ROUTINE          ;;;;;;;;;;;;;
                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                       ;  This routine keeps track of the current time and position of the motor.
                       ;  The upper word of information is provided by the timer overflow routine.

2400                             CSEG AT 2400H
2400 20CE              now_mode_1:       br      in_mode_1        ; used to save execution time for
2402 20C7              no_int1:          br      no_int           ; worst case loop

2404 F2                hsi_data_int:     pushf
2405 91400F                      orb     port1,#01000000B         ; set P1.6
2408 717F6D                      andb    ios1_bak,#01111111B      ; Clear ios1_bak.7
240B 90166D                      orb     ios1_bak,ios1
240E 376DF1                      jbc     ios1_bak,7,no_int1       ; If hsi is not triggered then
                                                                  ; jump to no_int
2411                   get_values:
2411 A00C28                      ld      timer_2,TIMER2
2414 5155066A                    andb    hsi_s0,HSI_STATUS,#01010101B
2418 A00440                      ld      time, HSI_TIME

241B 380FE2                      jbs     port1,0,now_mode_1       ; jump if in mode 1

241E                   In_mode_0:
241E 386A0B                      jbs     hsi_s0,0,a_rise
```

270658-A4

```
2421 3A6A2C                436             jbs     hsi_s0,2,a_fall
2424 3C6A4D                437             jbs     hsi_s0,4,b_rise
2427 3E6A5A                438             jbs     hsi_s0,6,b_fall
242A 2094                  439             br      no_cnt
                           440
242C A05658                441     a_rise: ld      last2_time,last1_time
242F A04056                442             ld      last1_time,time
2432 685840                443             sub     time,last2_time
2435 888240                444             cmp     time,min_hsi
2438 D906                  445             jh      tst_statr
                           446     ;set mode1-
243A 91010F                447             orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
243D B10515                448             ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
2440                       449     tst_statr:
2440 3E6B5B                450             jbs     last_stat,6,going_fwd
2443 3C6B67                451             jbs     last_stat,4,going_rev
2446 3A6B50                452             jbs     last_stat,2,change_dir
2449 98006B                453             cmpb    last_stat,zero
244C DF46                  454             je      first_time              ; first time in mode0
244E 27B2                  455             br      no_int1
                           456
2450 A05658                457     a_fall: ld      last2_time,last1_time
2453 A04056                458             ld      last1_time,time
2456 685840                459             sub     time,last2_time
2459 888240                460             cmp     time,min_hsi
245C D906                  461             jh      tst_statf
                           462     ;set mode1-
245E 91010F                463             orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
2461 B10515                464             ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
                           465     $EJECT
2464                       466     tst_statf:
2464 3C6B37                467             jbs     last_stat,4,going_fwd
2467 3E6B43                468             jbs     last_stat,6,going_rev
246A 386B2C                469             jbs     last_stat,0,change_dir
246D 98006B                470             cmpb    last_stat,zero
2470 DF22                  471             je      first_time              ; first time in mode0
2472 2057                  472             br      no_int
                           473
2474 386B27                474     b_rise: jbs     last_stat,0,going_fwd
2477 3A6B33                475             jbs     last_stat,2,going_rev
247A 3E6B1C                476             jbs     last_stat,6,change_dir
247D 98006B                477             cmpb    last_stat,zero
2480 DF12                  478             je      first_time              ; first time in mode0
2482 2047                  479             br      no_int
                           480
2484 3A6B17                481     b_fall: jbs     last_stat,2,going_fwd
2487 386B23                482             jbs     last_stat,0,going_rev
248A 3C6B0C                483             jbs     last_stat,4,change_dir
248D 98006B                484             cmpb    last_stat,zero
2490 DF02                  485             je      first_time              ; first time in mode0
```

270658-A5

```
2492 2037                486             br      no_int
                         487
2494                     488 first_time:
2494 C46B6A              489             stb     hsi_s0,last_stat
2497 2072                490             br      done_chk        ; add delta position
                         491
                         492
2499                     493 change_dir:
2499 1268                494             notb    direct
249B 30680F              495 no_inc:     jbc     direct,0,going_rev
                         496
249E                     497 going_fwd:
249E 914010              498             orb     PORT2,#01000000B        ; set P2.6
24A1 B10168              499             ldb     direct,#01              ; direction = forward
24A4 65010030            500             add     position,#01
24A8 A40032              501             addc    position+2,zero
24AB 200D                502             br      st_stat
24AD                     503 going_rev:
24AD 71BF10              504             andb    PORT2,#10111111B        ; clear P2.6
24B0 B10068              505             ldb     direct,#00              ; direction = reverse
24B3 69010030            506             sub     position,#01
24B7 A80032              507             subc    position+2,zero
                         508
24BA                     509 st_stat:
24BA C46B6A              510             stb     hsi_s0,last_stat
24BD                     511 load_lasts:
24BD A0282C              512             ld      tmr2_old,timer_2
24C0 717F6D              513 no_cnt:     andb    ios1_bak,#01111111B     ; clr bit 7
24C3 90166D              514             orb     ios1_bak,ios1
24C6 376D02              515             jbc     ios1_bak,7,no_int
24C9 2746                516 again:      br      get_values
                         517
24CB 71BF0F              518 no_int:     andb    port1,#10111111B        ; Clear P1.6
24CE F3                  519             popf
24CF F0                  520             ret     ; end of hsi_data interrupt routine
                         521             ; Routine for mode 1 follows and then returns to "load_lasts"
                         522 $EJECT
                         523
                         524
24D0                     525 In_mode_1:          ; mode 1 HSI routine
                         526
24D0 51506A5C            527             andb    tmp1,hsi_s0,#01010000B
24D4 D7EA                528             jne     no_cnt
24D6                     529 cmp_time:                           ; Procedure which sets mode 1 also
                         530                                     ; sets times to pass the tests
24D6 A05658              531             ld      last2_time,last1_time
24D9 A04056              532             ld      last1_time,time
                         533
24DC 4858405C            534 cmp1:       sub     tmp1,time,last2_time
24E0 88845C              535             cmp     tmp1,min_hsi1
```

270658-A6

```
24E3 D914                            jh      check_max_time

24E5                    set_mode_2:
24E5 91020F                          orb     Port1,#00000010B         ; Set P1.1 (in mode 2)
24E8 B10015                          ldb     IOC0,#00000000B          ; Disable all HSI
24EB A00400             mt_hsi:      ld      zero,hsi_time            ; empty the hsi fifo
24EE 717F6D                          andb    ios1_bak,#01111111B              ; clear bit 7
24F1 90166D                          orb     ios1_bak,ios1
24F4 3F6DF4                          jbs     ios1_bak,7,mt_hsi        ; If hsi is triggered then clear hsi
24F7 2012                            br      done_chk

24F9                    check_max_time:
24F9 4858405C                        sub     tmp1,time,last2_time
24FD 88865C                          cmp     tmp1,max_hsi1            ; max_hsi = addition to min_hsi for
                                                                      ; total time
2500 D109                            jnh     done_chk

2502                    set_mode_0:
2502 71FC0F                          andb    Port1,#11111100B         ; clear P1.0,1  set mode 00
2505 B15515                          ldb     IOC0,#01010101B          ; Enable all HSI
2508 B0006B                          ldb     last_stat,zero

250B                    done_chk:
250B 482C283C                        sub     delta_p,timer_2,tmr2_old         ; get timer2 count difference
250F 306808                          jbc     direct,0,add_rev
2512                    add_fwd:
2512 643C30                          add     position,delta_p
2515 A40032                          addc    position+2,zero
2518 27A3                            br      load_lasts
251A                    add_rev:
251A 683C30                          sub     position,delta_p
251D A80032                          subc    position+2,zero
2520 279B                            br      load_lasts

                        $eject
                        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        ;;;;;;                   SOFTWARE TIMER ROUTINE 1                  ;;;;;;;;;;;
                        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2600                    CSEG AT 2600H

2600                    swt1_expired:

2600 F2                              pushf
2601 91800F                          orb     port1,#10000000B         ; set port1.7

2604 B10D08                          ldb     int_mask,#00001101B      ; enable HSI, Tovf, HSO

2607 B13906                          ldb     HSO_COMMAND,#39H
260A 447E0A04                        add     HSO_TIME,TIMER1,swt1_dly
```

270658–A7

```
                        586
260E A0464A             587             ld      time_err+2,des_time+2   ; Calculate time & position error
2611 A0363A             588             ld      pos_err+2,des_pos+2
2614 48404448           589             sub     time_err,des_time,time          ; values are set
2618 A8424A             590             subc    time_err+2,time+2
261B 48303438           591             sub     pos_err,des_pos,position
261F A8323A             592             subc    pos_err+2,position+2
                        593
2622 FB                 594             EI
                        595
2623 48484C52           596             sub     time_delta,last_time_err,time_err
2627 A0484C             597             ld      last_time_err,time_err
                        598
262A 48384E50           599             sub     pos_delta,last_pos_err,pos_err
262E A0384E             600             ld      last_pos_err,pos_err
                        601
                        602             ;;;;;   Time_err = Desired time to finish - current time
                        603             ;;;;;   Pos_err  = Desired position to finish - current position
                        604             ;;;;;   Pos_delta  = Last position error - Curent position error
                        605             ;;;;;   Time_delta = Last time error - Current time error
                        606             ;;;;;     note that errors should get smaller so deltas will be
                        607             ;;;;;     positive for forward motion (time is always forward)
                        608
                        609
2631                    610     chk_dir:
2631 B8003A             611             cmp     pos_err+2,zero
2634 D60D               612             jge     go_forward
                        613
2636                    614     go_backward:
2636 0338               615             neg     pos_err         ; Pos_err = ABS VAL (pos_err)
2638 B10069             616             ldb     pwm_dir,#00h
263B B9FFFF3A           617             cmp     pos_err+2,#0ffffH
263F D70A               618             jne     ld_max
2641 200D               619             br      chk_brk
                        620
2643                    621     go_forward:
2643 B10169             622             ldb     pwm_dir,#01H
2646 B8003A             623             cmp     pos_err+2,zero
2649 DF05               624             je      chk_brk
                        625     $EJECT
                        626
264B B0706C             627     ld_max: ldb     pwm_pwr,max_pwr
264E 2051               628             br      chk_sanity
                        629
2650                    630     Chk_brk:                        ;  Position_Error now = ABS(pos_err)
2650 B87A38             631             cmp     pos_err,pos_pnt
2653 D11E               632             jnh     hold_position   ; position_error<position_control_point
2655 B87B38             633             cmp     pos_err,brk_pnt
```

270658-A8

```
2658 D9F1                634             jh      ld_max          ; position_error>brake_point
                         635
265A                     636 braking:
265A 880050              637             cmp     pos_delta,zero
265D D602                638             jge     chk_delta
265F 0350                639             neg     pos_delta
2661                     640 chk_delta:
2661 887650              641             cmp     pos_delta,vel_pnt       ; velocity = pos_delta/sample_time
2664 D10D                642             jnh     hold_position           ; jmp if  ABS(velocity) < vel_pnt
                         643
2666 B0726C              644 brake:      ldb     pwm_pwr,max_brk
2669 B06824              645             ldb     tmp,direct              ; If braking apply power in opposite
266C 1224                646             notb    tmp                     ; direction of current motion
266E B02469              647             ldb     pwm_dir,tmp
                         648
2671 2030                649             br      ld_pwr
                         650
2673                     651 Hold_position:                      ; position hold mode
2673 89020038            652             cmp     pos_err,#02
2677 D906                653             jh      calc_out        ; if position error < 2 then turn off power
2679 0126                654             clr     tmp+2
267B 015A                655             clr     boost
267D 201F                656             BR      output
                         657
267F                     658 calc_out:
267F 5DFF7424            659             mulub   tmp,max_hold,#255
2683 6C3824              660             mulu    tmp,pos_err             ; Tmp = pos_err * max_hold
2686 880050              661             cmp     pos_delta,zero
2689 D709                662             jne     no_bst
268B 6504005A            663             add     boost,#04               ; Boost is integral control
268F 645A26              664             add     tmp+2,boost             ; TMP+2 = MSB(pos_err*max_hold)
2692 2002                665             br      ck_max
2694 015A                666 no_bst:     clr     boost
2696 887426              667 ck_max:     cmp     tmp+2,max_hold
2699 D103                668             jnh     output
269B A07426              669 maxed:      ld      tmp+2,max_hold
269E B0266C              670 output:     ldb     pwm_pwr,tmp+2
                         671
                         672
26A1                     673 chk_sanity:
26A1 2000                674             br      ld_pwr
                         675 ;;
                         676 ;;
                         677 $EJECT
                         678
26A3                     679 ld_pwr:
26A3 B06C64              680             ldb     rpwr,pwm_pwr
26A6 1264                681             notb    rpwr
26A8 38690A              682             jbs     pwm_dir,0,p2fwd
                         683
```

270658-A9

```
26AB FA              p2bkwd: DI
26AC 717F10                  andb    port2,#01111111B          ; clear P2.7
26AF B06417                  ldb     pwm_control,rpwr
26B2 FB                      EI
26B3 2008                    br      pwrset
26B5 FA              p2fwd:  DI
26B6 918010                  orb     port2,#10000000B          ; set P2.7
26B9 B06417                  ldb     pwm_control,rpwr
26BC FB                      EI
26BD                 pwrset:
26BD 88004A                  cmp     time_err+2,zero   ; do pos_table when err is negative
26C0 D225                    jgt     end_p
                     ;;;     br      end_p
26C2 89202962                cmp     nxt_pos,#(32+pos_table)
26C6 DE06                    jlt     get_vals                  ; jump if lower
26C8 A1002962                ld      nxt_pos,#pos_table
26CC 0142                    clr     time+2
26CE                 get_vals:
26CE A26334                  ld      des_pos,[nxt_pos]+
26D1 A26336                  ld      des_pos+2,[nxt_pos]+
26D4 A26346                  ld      des_time+2,[nxt_pos]+
26D7 A26370                  ld      max_pwr,[nxt_pos]+
26DA A07072                  ld      max_brk,max_pwr
26DD 646034                  add     des_pos,offset
26E0 A40036                  addc    des_pos+2,zero
26E3 4830344E                sub     last_pos_err,des_pos,position
26E7 717F0F          end_p:  andb    port1,#01111111B          ; clear P1.7
26EA F3                      popf
26EB F0                      ret
                     $EJECT
                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                     ;;;;;;;;;;;;;;;;;;;;;;;;        main  program        ;;;;;;;;;;;;;;;;;;;;;;;;
                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2800                         CSEG at 2800H
2800                 MAIN_PROG:
2800 90166D                  orb     ios1_bak,ios1
2803 366D09                  jbc     ios1_bak,6,control
2806 71BF6D                  andb    ios1_bak,#10111111B       ; clear ios1_bak.6
2809 95100F                  xorb    Port1,#00010000B          ; Compl Bit P1.4
280C EFF5FB                  call    HSI_DATA_INT              ; prevent lockup
```

270658–B0

```
280F                    734   control:
280F 912D08             735           orb     int_mask,#00101101B     ; enable hsi, hso, swt, tovf interrupts
2812 FD                 736           nop
2813 FD                 737           nop
2814 FD                 738           nop
2815 E06FFD             739           djnz    main_dly,$
2818 FD                 740           nop
2819 95080F             741           xorb    port1,#00001000B        ; compliment p1.3
281C 27E2               742           BR      MAIN_PROG
                        743
                        744
2900                    745           CSEG AT 2900H
                        746
2900                    747   pos_table:
                        748
2900 00000000           749           dcl     00000000H       ; position 0
2904 20008000           750           dcw     0020H, 0080H    ; next time, power
2908 00C00000           751           dcl     0000c000H       ; position 1
290C 40004000           752           dcw     0040H, 0040H    ; next time, power
2910 00000000           753           dcl     00000000H       ; position 2
2914 6000C000           754           dcw     0060H, 00c0H    ; next time, power
2918 0080FFFF           755           dcl     0FFFF8000H      ; position 3
291C 80008000           756           dcw     0080H, 0080H    ; next time, power
                        757
2920 00080000           758           dcl     00000800H       ; position 4
2924 58008000           759           dcw     0058H, 0080H    ; next time, power
2928 00300000           760           dcl     00003000H       ; position 5
292C 7000FF00           761           dcw     0070H, 00ffH    ; next time, power
2930 00000000           762           dcl     00000000H       ; position 6
2934 9000F000           763           dcw     0090H, 00f0H    ; next time, power
2938 00000000           764           dcl     00000000H       ; position 7
293C 9100F000           765           dcw     0091H, 00f0H    ; next time, power
                        766
                        767
2940                    768           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270658-B1

```
MACRO ASSEMBLY COMPLETED.  NO ERROR(S) FOUND

5400                    END

5400                    CSEG AT 5400H

5400                    bar_table:

5400 00000000           DW      0,0,0,0

5408 00000000           DW      0,0,0,0

5410 00000000           DW      0,0,0,0

5418 00000000           DW      0,0,0,0

5420 00000000           DW      0,0,0,0

5428 00000000           DW      0,0,0,0

5430 00000000           DW      0,0,0,0

5438 00000000           DW      0,0,0,0

5440 00000000           DW      0,0,0,0

5448 00000000           DW      0,0,0,0

5450 00000000           DW      0,0,0,0

5458 00000000           DW      0,0,0,0

5460 00000000           DW      0,0,0,0

5468 00000000           DW      0,0,0,0

5470 00000000           DW      0,0,0,0

5478 00000000           DW      0,0,0,0

5480 00000000           DW      0,0,0,0

5488 00000000           DW      0,0,0,0

5490 00000000           DW      0,0,0,0

5498 00000000           DW      0,0,0,0

54A0 00000000           DW      0,0,0,0

54A8 00000000           DW      0,0,0,0

54B0 00000000           DW      0,0,0,0

54B8 00000000           DW      0,0,0,0

54C0 00000000           DW      0,0,0,0

54C8 00000000           DW      0,0,0,0

54D0 00000000           DW      0,0,0,0

54D8 00000000           DW      0,0,0,0

54E0 00000000           DW      0,0,0,0

54E8 00000000           DW      0,0,0,0

54F0 00000000           DW      0,0,0,0

54F8 00000000           DW      0,0,0,0

5500                    END
```

MCS®-96 Data Sheets

6

MSC®-96
809XBH, 839XBH, 879XBH
ADVANCED 16-BIT MICROCONTROLLER WITH 8- OR 16-BIT EXTERNAL BUS

- **879XBH: an 809XBH with 8K Bytes of On-Chip EPROM**
- **839XBH: an 809XBH with 8K Bytes of On-Chip ROM**

- **232 Byte Register File**
- **Register-to-Register Architecture**
- **10-Bit A/D Converter with S/H**
- **Five 8-Bit I/O Ports**
- **20 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **ROM/EPROM Lock**
- **Run-Time Programmable EPROM**
- **High Speed I/O Subsystem**
- **Full Duplex Serial Port**
- **Dedicated Baud Rate Generator**
- **6.25 μs 16 x 16 Multiply**
- **6.25 μs 32/16 Divide**
- **16-Bit Watchdog Timer**
- **Four 16-Bit Software Timers**
- **Two 16-Bit Counter/Timers**

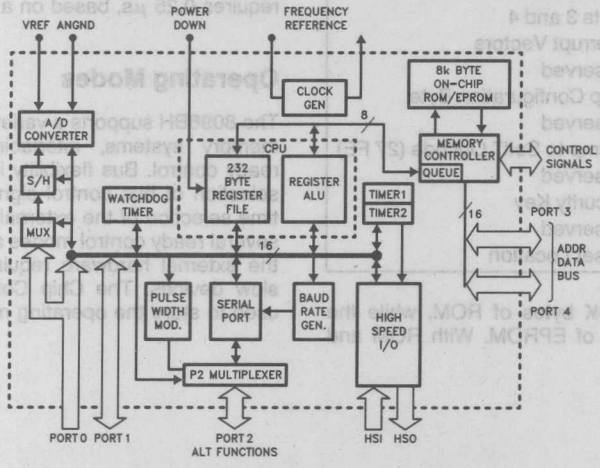
The MCS-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. The MCS-96 family members produced using Intel's HMOS-III process are described in this data sheet.

The CPU supports bit, byte, and word operations. Thirty-two bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096BH can do a 16-bit addition in 1.0 μs and a 16 x 16-bit multiply or 32/16 divide in 6.25 μs. Instruction execution times average 1 to 2 μs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform software timer functions. Up to four 16-bit software timers can be in operation at once.

The on-chip A/D converter includes a Sample and Hold, and converts up to 8 multiplexed analog input channels to 10-bit digital values. With a 12 MHz crystal, each conversion takes 22 μs. This feature is only available on the 8X95BHs and 8X97BHs, with the 8X95BHs having 4 multiplexed analog inputs.

Also provided on-chip are a serial port, a Watchdog Timer, and a pulse-width modulated output signal.



270090-50

Figure 1. MCS®-96 Block Diagram

October 1988
Order Number: 270090-004

FUNCTIONAL OVERVIEW

The following section is an overview of the 8X9XBH devices, generally referred to as the 8096BH. Additional information is available in the Embedded Controller Handbook, order number 210918.

CPU Architecture

The 8096BH uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems).

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down via a separate power down pin (V_{PD}).

Outside of the Register File, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27 FE)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

The 839XBH carries 8K bytes of ROM, while the 879XBH has 8K bytes of EPROM. With ROM and EPROM parts, the internal program memory occupies addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip memory if the $\overline{EA}$ pin is externally held at 5V. If the $\overline{EA}$ pin is at 0V, these addresses access off-chip memory. On the 879XBH parts, holding $\overline{EA}$ at +12.75V puts the part in Programming Mode, which is described in the EPROM Characteristics Section of this data sheet.

A memory map for the MCS-96 product family is shown in Figure 2.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096BH is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

- 6.25 μs Multiply and Divide
- Multiple Shift Instruction
- 3 Operand Instructions
- Normalize Instruction
- Software Reset Instruction

All operations on the 8096BH take place in a set number of "State Times." The 8096BH uses a three phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 μs, based on a T_{osc} of 83 ns.

Operating Modes

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and run-time selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register is used to store the operating mode information.

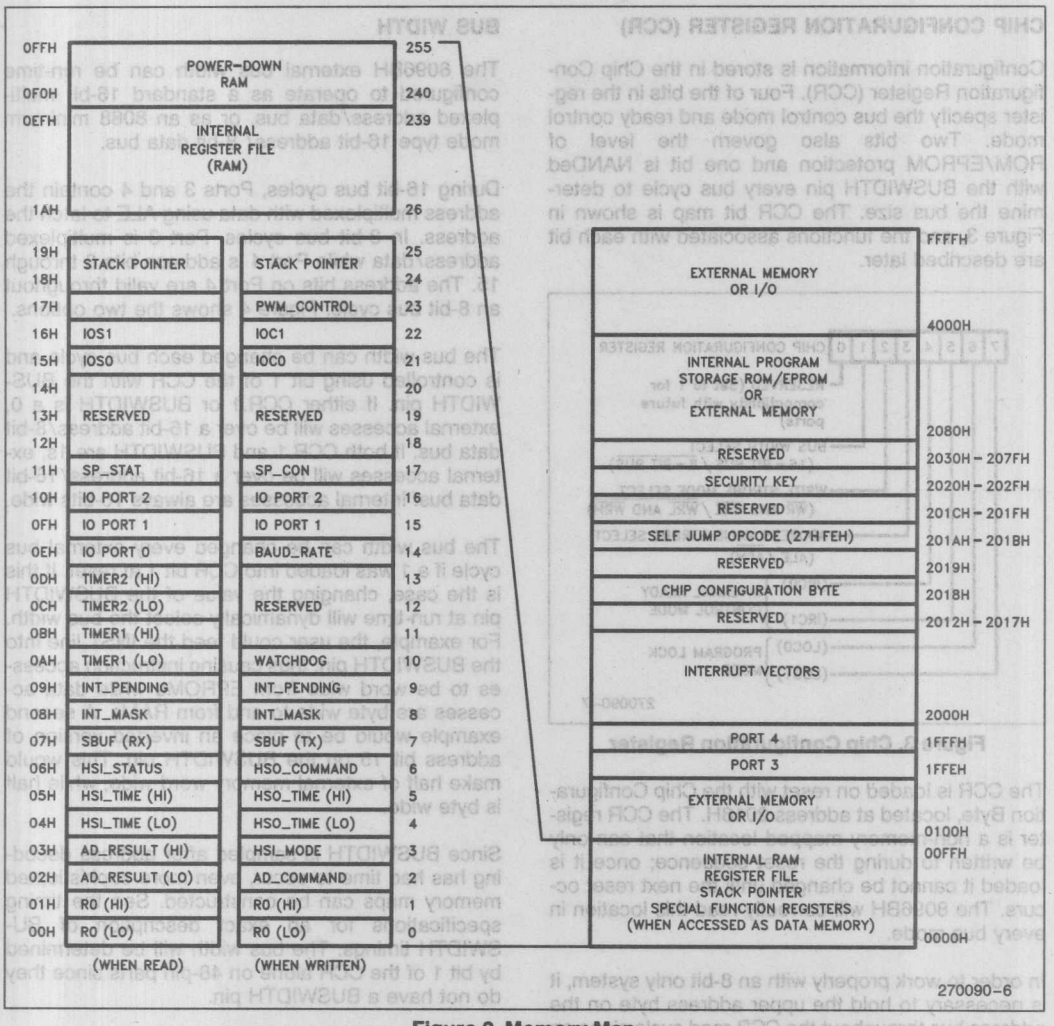


Figure 2. Memory Map

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 3, and the functions associated with each bit are described later.

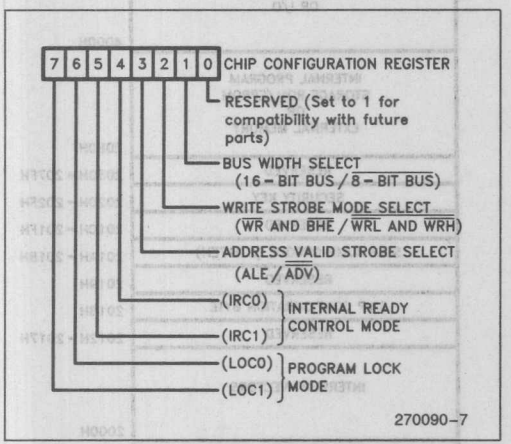


Figure 3. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

In order to work properly with an 8-bit only system, it is necessary to hold the upper address byte on the address bus throughout the CCB read cycle since an address latch may not be present. However, in a 16-bit system, the 8X9XBH must float the high half of the bus to avoid contention with the high data byte during the CCB read. To accomplish a correct read on either 8- or 16-bit buses, the upper address lines are current sensed (during CCB read only) and will be floated if a current of approximately 1 mA or more is detected, indicating a bus contention.

If the $\overline{EA}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{EA}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{EA}$ is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming Mode is described in the EPROM Characteristics Section.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8088 minimum mode type 16-bit address/ 8-bit data bus.

During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 4 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more sophisticated memory maps can be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one).

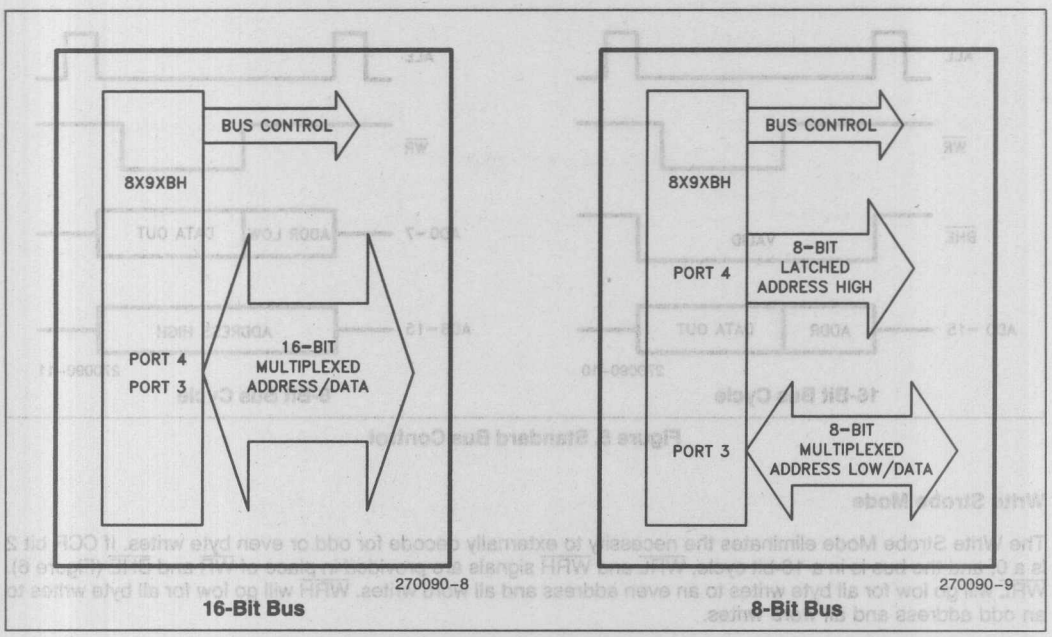


Figure 4. Bus Width Options

BUS CONTROL

The 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{WR}$, $\overline{BHE}$ and ALE are provided (Figure 5). $\overline{WR}$ will come out for every write. $\overline{BHE}$ will be valid throughout the bus cycle and can be combined with $\overline{WR}$ and address line 0 to form $\overline{WRL}$ and $\overline{WRH}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address.

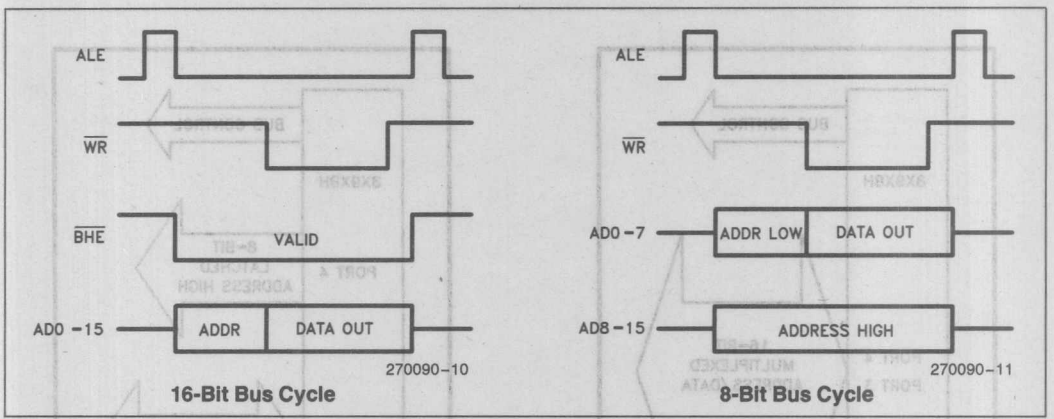


Figure 5. Standard Bus Control

Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{WRL}$ and $\overline{WRH}$ signals are provided in place of $\overline{WR}$ and $\overline{BHE}$ (Figure 6). $\overline{WRL}$ will go low for all byte writes to an even address and all word writes. $\overline{WRH}$ will go low for all byte writes to an odd address and all word writes.

In an 8-bit bus cycle $\overline{WRL}$ will go active for all writes.

A unique ability of the bus controller is to utilize the CCR to select at reset time the width of the $\overline{WR}$ signal by changing the position of the falling edge relative to the memory cycle. Clearing bit 2 of the CCR to 0 will enable a shorter $\overline{WR}$ width. This is useful when interfacing to devices that latch data on the falling edge of $\overline{WR}$.

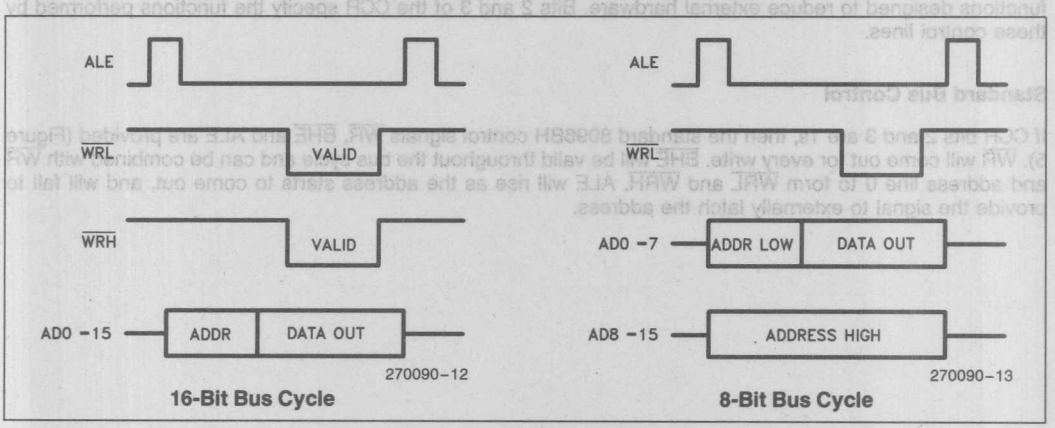


Figure 6. Write Strobe Mode

Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid Strobe is provided in the place of ALE (Figure 7). When the Address Valid Mode is selected, $\overline{ADV}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.

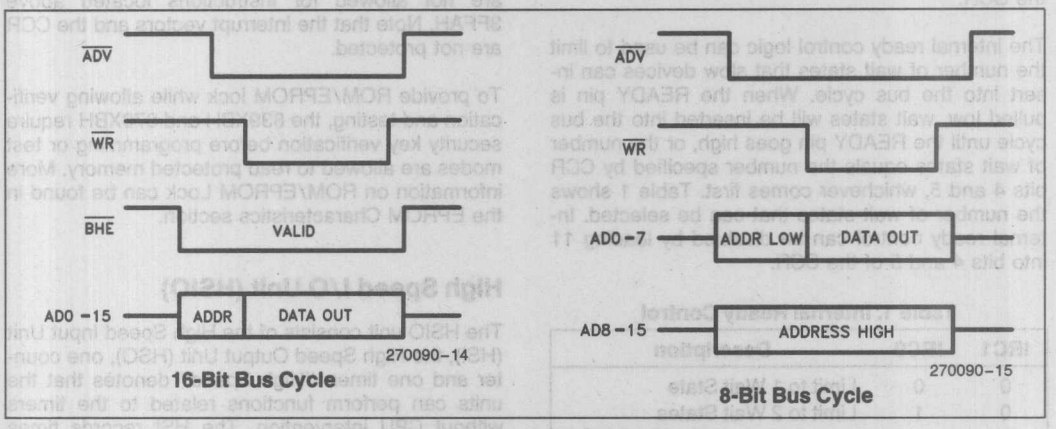


Figure 7. Address Valid Strobe Mode

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid Strobe and the Write Strobes will be provided for bus control. Figure 8 shows these signals.

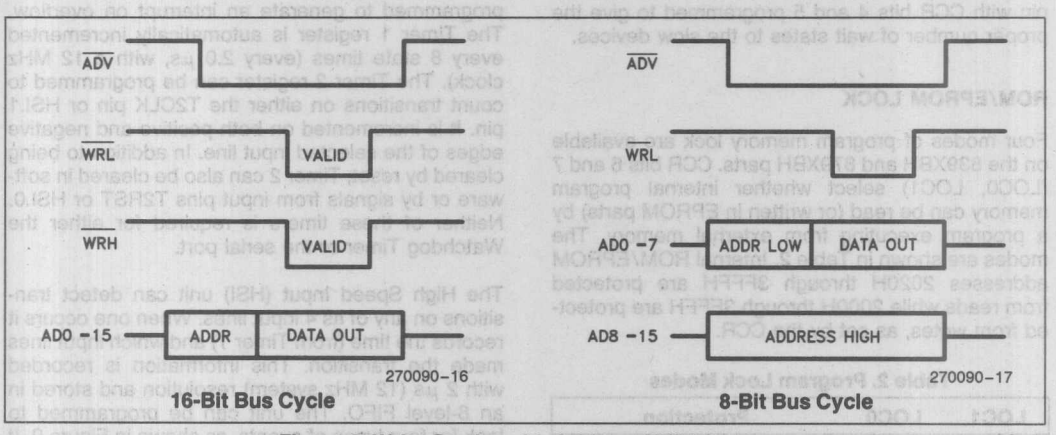


Figure 8. Write Strobe with Address Valid Strobe

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the proper number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. Note that the interrupt vectors and the CCR are not protected.

To provide ROM/EPROM lock while allowing verification and testing, the 839XBH and 879XBH require security key verification before programming or test modes are allowed to read protected memory. More information on ROM/EPROM Lock can be found in the EPROM Characteristics section.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at pre-programmed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 μs, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HSI.0. Neither of these timers is required for either the Watchdog Timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines made the transition. This information is recorded with 2 μs (12 MHz system) resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 9. It can activate the HSI Data Available interrupt either when the Holding Register is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

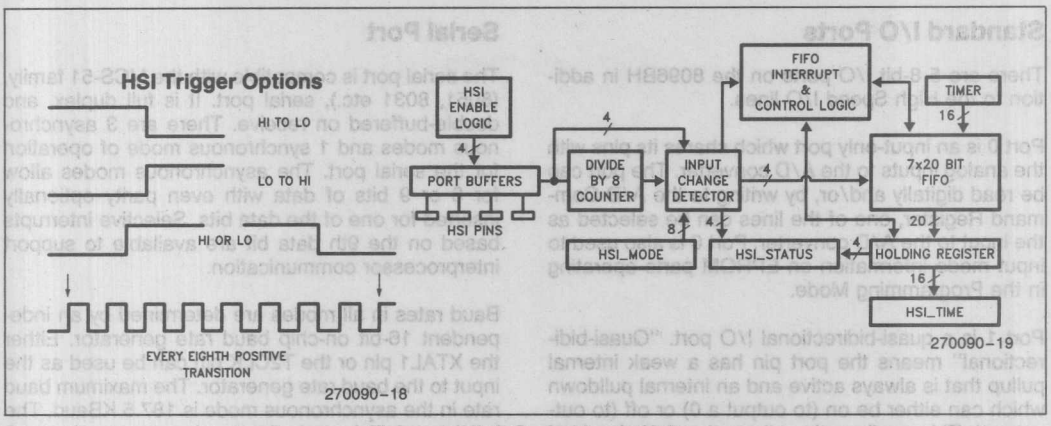


Figure 9. High Speed Input Unit

The High Speed Output (HSO) unit is shown in Figure 10. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timer flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

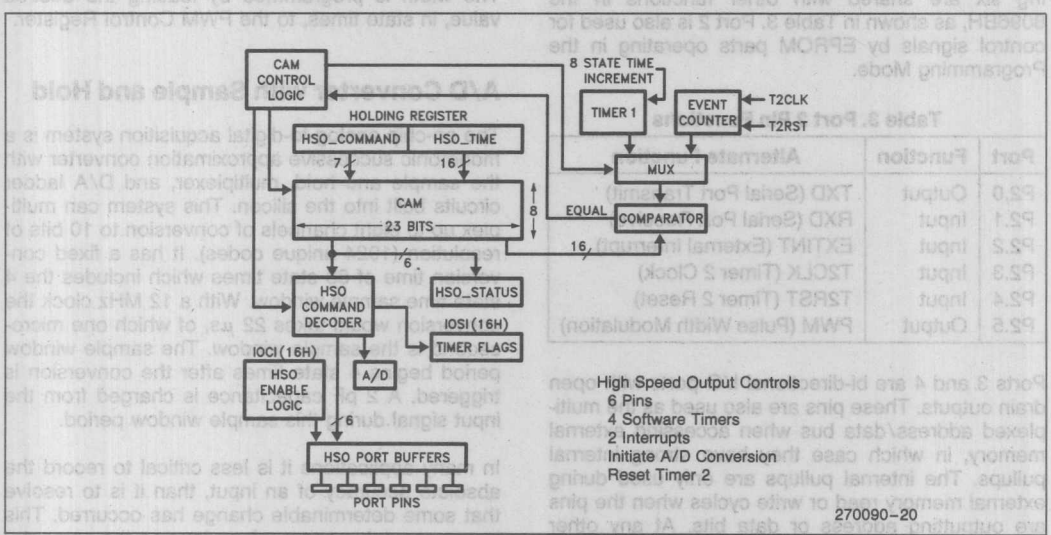


Figure 10. High Speed Output Unit

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096BH in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D converter. Port 0 is also used to input mode information on EPROM parts operating in the Programming Mode.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Two of the pins (P2.6, 2.7) are quasi-bidirectional while the remaining six are shared with other functions in the 8096BH, as shown in Table 3. Port 2 is also used for control signals by EPROM parts operating in the Programming Mode.

Table 3. Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled. When used as a system bus, Ports 3 and 4 can be configured to be either a multiplexed 16-bit address/data bus or a multiplexed 16-bit address/ 8-bit data bus. EPROM parts also use Ports 3 and 4 to pass programming commands, addresses, data and status.

Serial Port

The serial port is compatible with the MCS-51 family, (8051, 8031 etc.), serial port. It is full duplex, and double-buffered on receive. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud. The maximum baud rate in the synchronous mode is 1.5 MBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter with Sample and Hold

The on-chip analog-to-digital acquisition system is a monotonic successive approximation converter with the sample and hold, multiplexer, and D/A ladder circuits built into the silicon. This system can multiplex up to eight channels of conversion to 10 bits of resolution (1024 unique codes). It has a fixed conversion time of 88 state times which includes the 4 state time sample window. With a 12 MHz clock the conversion would takes 22 μs, of which one microsecond is the sample window. The sample window period begins 4 state times after the conversion is triggered. A 2 pF capacitance is charged from the input signal during this sample window period.

In many applications it is less critical to record the absolute accuracy of an input, than it is to resolve that some determinable change has occurred. This is an acceptable approach as long as the converter is guaranteed to be monotonic and has no missing

codes, as is the case for the 8X9XBH. This means that increasing input voltages produce adjacent and unique output codes that are also increasing. Decreasing input voltages are guaranteed to produce adjacent and unique output codes that are also decreasing. There exists on the 8X9XBH for each 10-bit output code a unique input voltage range that produces that code only, with a repeatability of typically ±0.25 LSB's (1.5 mV).

The 8X9XBH datasheet guarantees that the maximum Differential Non-Linearity will be 2 analog LSB's, or 10 mV (the minimum is zero). Differential non-linearity specifies the **maximum difference** between the actual code widths seen in a converter and what those code widths would be in an ideal (perfect) converter. In the 8X9XBH 10-bit converter, the code widths are ideally 5 mV (5.12 V_{REF}/1024). If such a converter is specified to have a maximum Differential Non-Linearity of 10 mV, then the maximum code width will be no greater than 10 mV larger than ideal, or 15 mV. This indicates to the user how much the input voltage may have changed under worst case conditions to produce a one count change in a particular 10-bit conversion. Due to the fact that the 8X9XBH converter has no missing codes, the minimum code width will always be greater than zero. The differential non-linearity error on a particular code width is compensated for by other code widths in the transfer function such that 1024 unique steps occur. The actual code widths in the 8X9XBH converter typically vary from about 2.5 mV to 7.5 mV.

The analog input must be in the range of zero to V_{REF} (nominally, V_{REF} = 5V). This input can be selected from 8 analog inputs which connect to the same pins as PORT 0. A conversion can be initiated either by setting the control bit in the A/D Command Register (Address 02Hex), or by programming the High Speed Output CAM to trigger the conversion at some specified time with sampling intervals occurring accurate to ±50 ns. (See AP 406, "MCS-96 Analog Acquisition Primer" for further information.)

Interrupts

The 8096BH has 20 interrupt sources which vector through 8 interrupt vectors. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. If it is to be serviced, the CPU pushes the current Program Counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 11.

The maximum transition speed of the interrupt inputs is limited to one transition per state time (250 ns at 12 MHz). Since interrupt recognition is based up on a zero to one transition on the pin, a normally high signal must go low for one state time, and then transition high in a subsequent state time. A normally low input signal must go high for one state time, and not return low until a subsequent state time.

Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 11. Interrupt Vectors

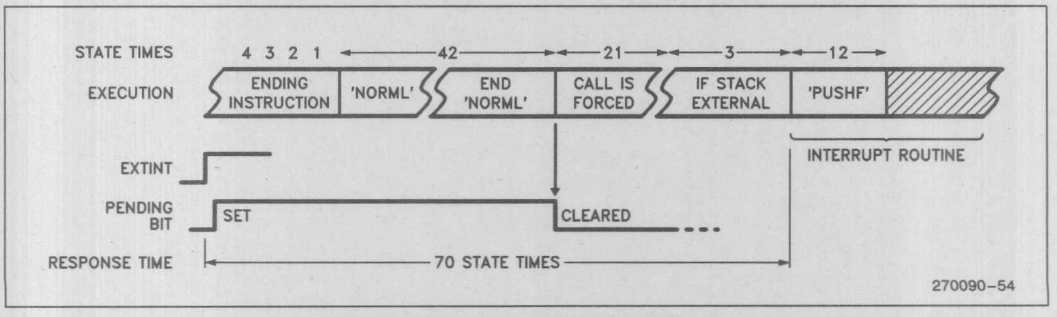


Figure 11A. Interrupt Response Time

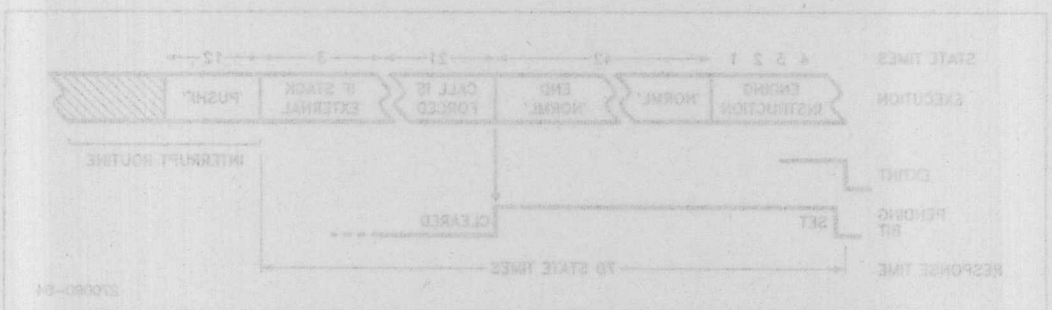
[illegible] the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the Register File. This feature of the architecture provides for very fast context switching. While the 8096BH has a single priority level in the sense that any interrupt may itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 11. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users software.

Watchdog Timer

The Watchdog Timer is a 16-bit counter which, once started, is incremented every state time. If not pulled down for two state times, causing the system to be reinitialized. In a 12 MHz system, the Watchdog Timer overflows after 16 ms.

This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$. Once the Watchdog Timer is started it cannot be turned off by software. The flip-flop which enables the Watchdog Timer has been designed to maintain its state through V_{CC} glitches to as low as 0V or as high as 7V for 1 μs to 1 ms.

To start the Watchdog Timer, or to clear it, one writes 1EH followed by 0E1H to the WDT address (000AH). The Watchdog cannot be stopped once it is started unless the system is reset.



PACKAGING

The 8096BH is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The 8096BH numbering system is shown in Figure 12. Figures 13–17 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

		Without A/D	With A/D
ROMless	48 Pin		C8095CH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC

Figure 12. The 8096BH Family Nomenclature

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}$/$\overline{WRH}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{WR}$/$\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5/$\overline{PDO}$/$\overline{SPROG}$	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{PROG}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

Figure 13. PGA, PLCC and LCC Function Pinouts

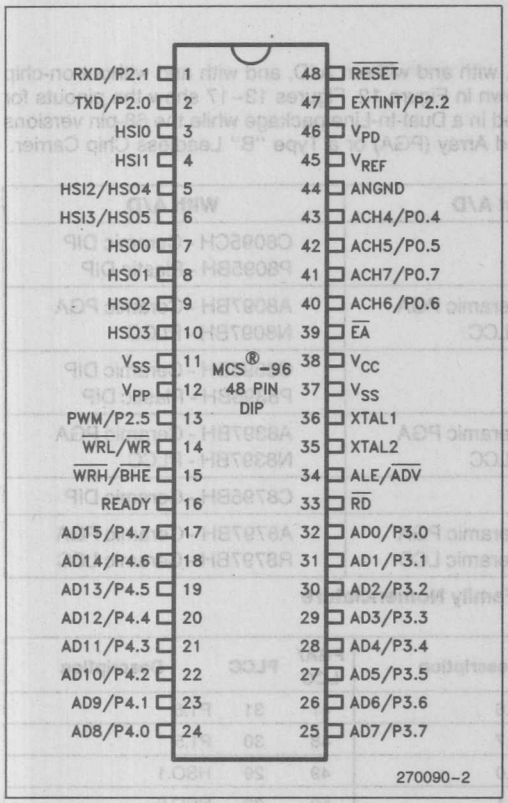


Figure 14. 48-Pin Package

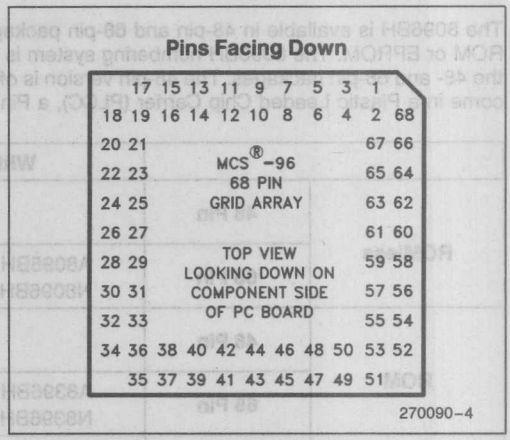


Figure 16. 68-Pin Package (Pin Grid Array - Top View)

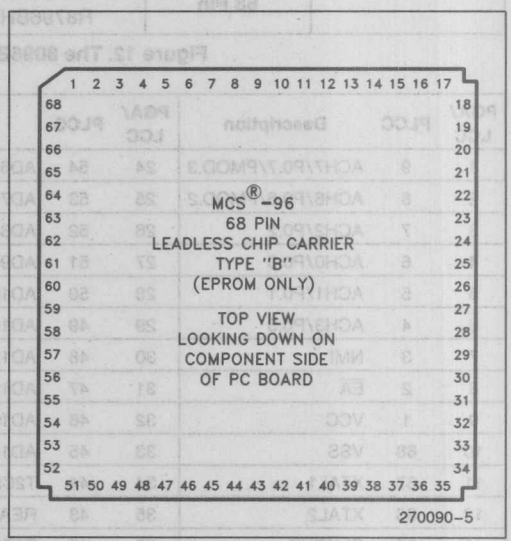


Figure 17. 68-Pin Package (LCC - Top View)

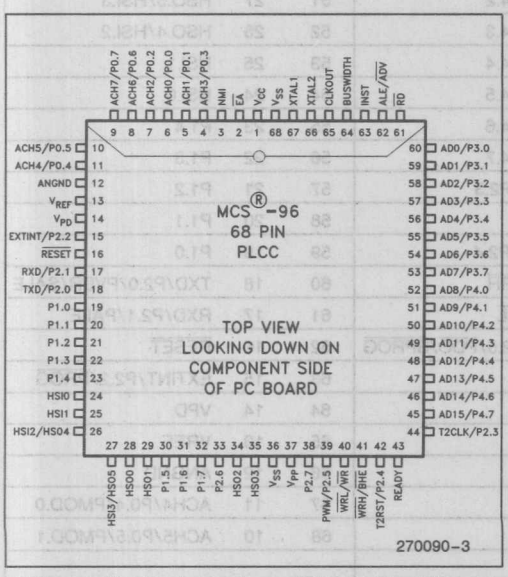


Figure 15. 68-Pin Package (PLCC - Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{RESET}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{RESET}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Programming voltage for the EPROM parts. It should be +12.75V for programming. This pin must be left floating in the application circuit.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.
$\overline{RESET}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{RESET}$ has an internal pullup.
BUSWIDTH	Input for bus width selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. If this pin is left unconnected, it will rise to V_{CC}.
NMI	A positive transition causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{EA}$ = +12.5V causes execution to begin in the Programming Mode. $\overline{EA}$ has an internal pulldown, so it goes to 0 unless driven otherwise.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.
$\overline{WR}$/$\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}$/$\overline{WRL}$ is activated only during external memory writes.
$\overline{BHE}$/$\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the Programming Mode.

INSTRUCTION SET

The 8096BH instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the word address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED (LONG AND SHORT)—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the word address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096BH contains a zero register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096BH, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270090-45

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

1. Number of state times shown for internal/external operands.
2. The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
8. State times shown for 16-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT② NORMAL			INDIRECT② AUTO-INC.		INDEXED② SHORT			INDEXED② LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES②	BYTES	STATE① TIMES②
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270090-46

NOTES:
1. Number of state times shown for internal/external operands.
3. The assembler does not accept this mnemonic.
4. The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
5. State times for stack located internal/external.
8. State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.(8)

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.(8)

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	EO	3	5/9 STATE TIME (NOT TAKEN/TAKEN)(8)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES(8)
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST(6)	FF	1	166	SKIP	00	2	4

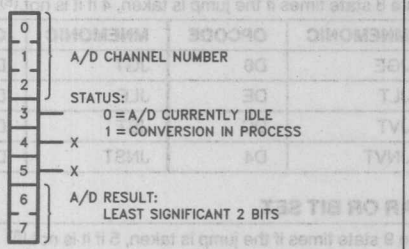
NORMALIZE

MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

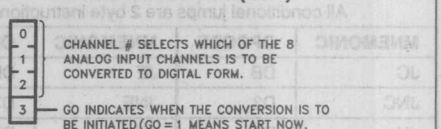
6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to $\overline{\text{RESET}}$, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

A/D Result LO (02H)



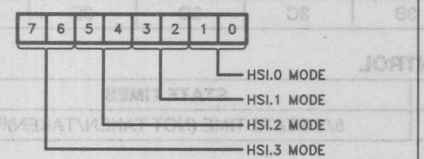
270090-21

A/D Command (02H)



270090-24

HSI_Mode (03H)

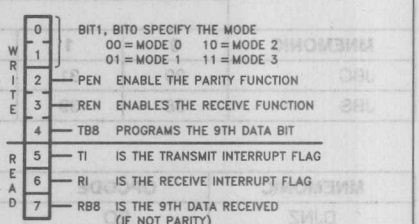


WHERE EACH 2 - BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

270090-22

SPCON/SPSTAT (11H)



270090-26

Baud Rate Calculations

Using XTAL1:

Mode 0: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$

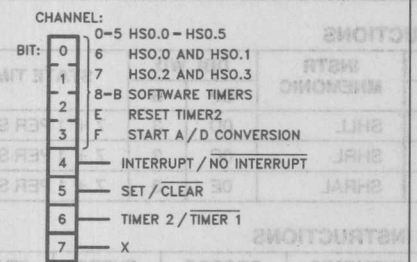
Using T2CLK:

Mode 0: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}; B \neq 0$

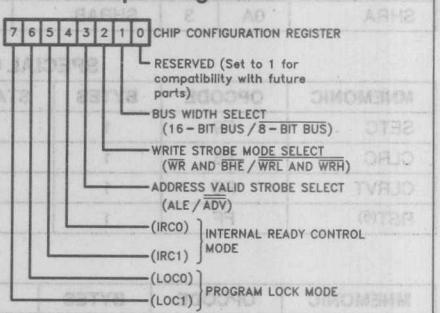
Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

HSO Command (06H)



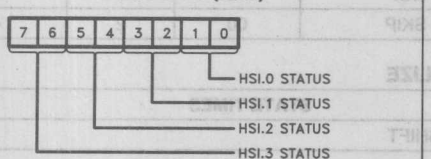
270090-23

Chip Configuration



270090-32

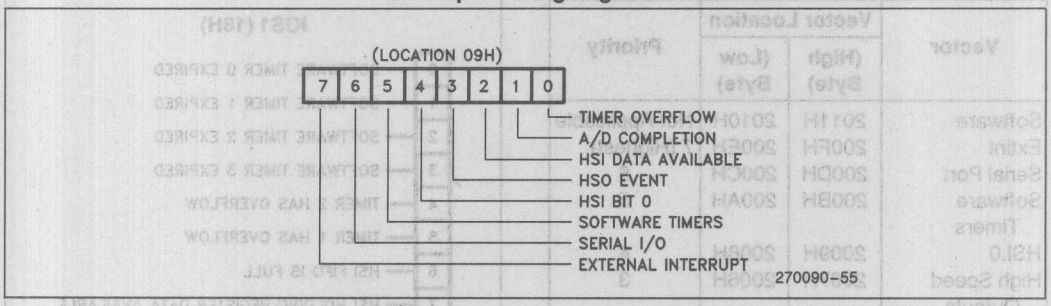
HSI_Status (06H)



WHERE FOR EACH 2 - BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270090-25

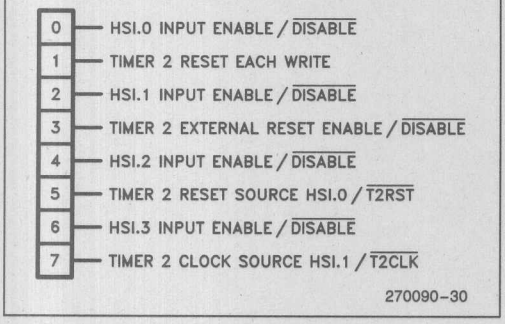
Interrupt Pending Register



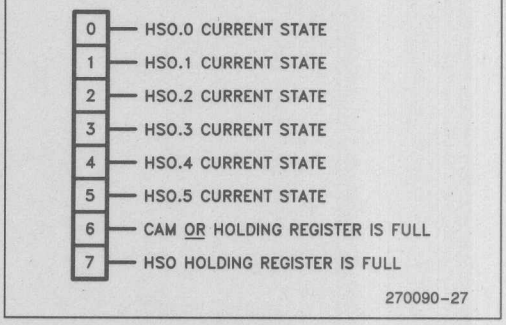
PSW Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

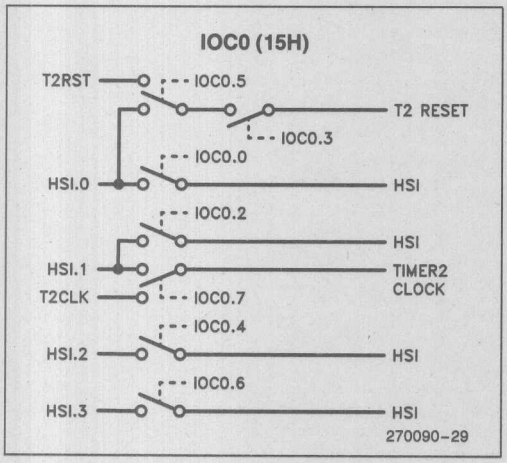
IOC0 (15H)



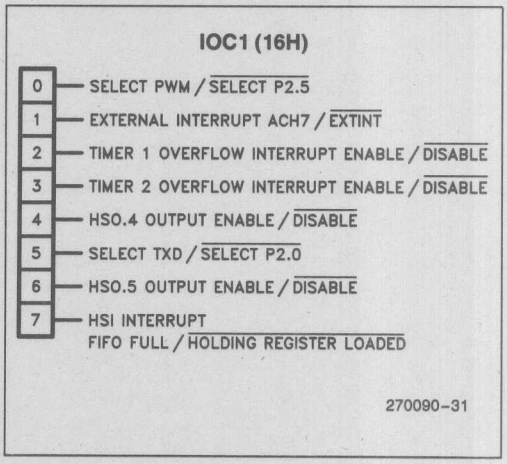
IOS0 (15H)



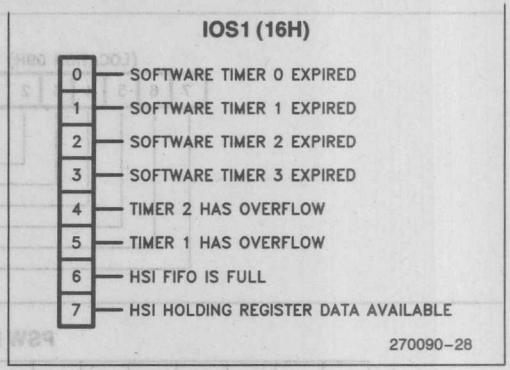
IOC0 (15H)



IOC1 (16H)



Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)



ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to +70°C

Storage Temperature . . . . . −40°C to +150°C

Voltage from $\overline{EA}$ or V_{PP} to V_{SS} or ANGND . . . . . −0.3V to +13.0V

Voltage from Any Other Pin to V_{SS} or ANGND . . . . . −0.3V to +7.0V*

Average Output Current from Any Pin . . . . . 10 mA

Power Dissipation 1.5W

*This includes V_{PP} on ROM and CPU only devices.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Test Conditions: V_{CC}, V_{REF}, V_{PD}, V_{PP}, V_{EA} = 5.0V ±0.5V; F_{OSC} = 6.0 MHz; T_A = 0°C to 70°C; V_{SS}, ANGND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current (0°C ≤ T_A ≤ 70°C)		240	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current (T_A = 70°C)		185	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		8	mA	
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling Hysteresis	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.2	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	µA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+3	µA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	µA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−125	µA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	−0.25	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY, BUSWIDTH		−50	µA	V_{IL} = 0.45V
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.45	V	I_{OL} = 0.8 mA (Note 1)
V_{OL1}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.75	V	I_{OL} = 2.0 mA (Notes 1, 2, 3)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{RESET}$ and Bus/Control Pins		0.45	V	I_{OL} = 2.0 mA (Notes 1, 2, 3, 4)

D.C. CHARACTERISTICS (Test Conditions: V_{CC}, V_{REF}, V_{PD}, V_{PP}, V_{EA} = 5.0V ±0.5V; F_{OSC} = 6.0 MHz; T_A = 0°C to 70°C; V_{SS}, ANGND = 0V) (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	I_{OH} = −20 µA (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	I_{OH} = −200 µA (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		µA	V_{OH} = 2.4V
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include TXD, RXD (Mode 0 only), PWM, and HSO pins. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins and Ports 3 and 4 when used as ports: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3. During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 8.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 15 mA.
   Total I_{OL} on Port 3 must not exceed 10 mA.
   Total I_{OL} on P2.5, P2.7, and Port 4 must not exceed 20 mA.
4. I_{OL} on HSO.X (X = 0, 4, 5) = 1.6 mA @ 0.5V.

A.C. CHARACTERISTICS V_{CC}, V_{PD} = 4.5 to 5.5V; T_A = 0°C to 70°C; f_{OSC} = 6.0 to 12.0 MHz

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 10 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−70	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−80	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−120	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−100	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−25	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −125	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +40		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −75	ns

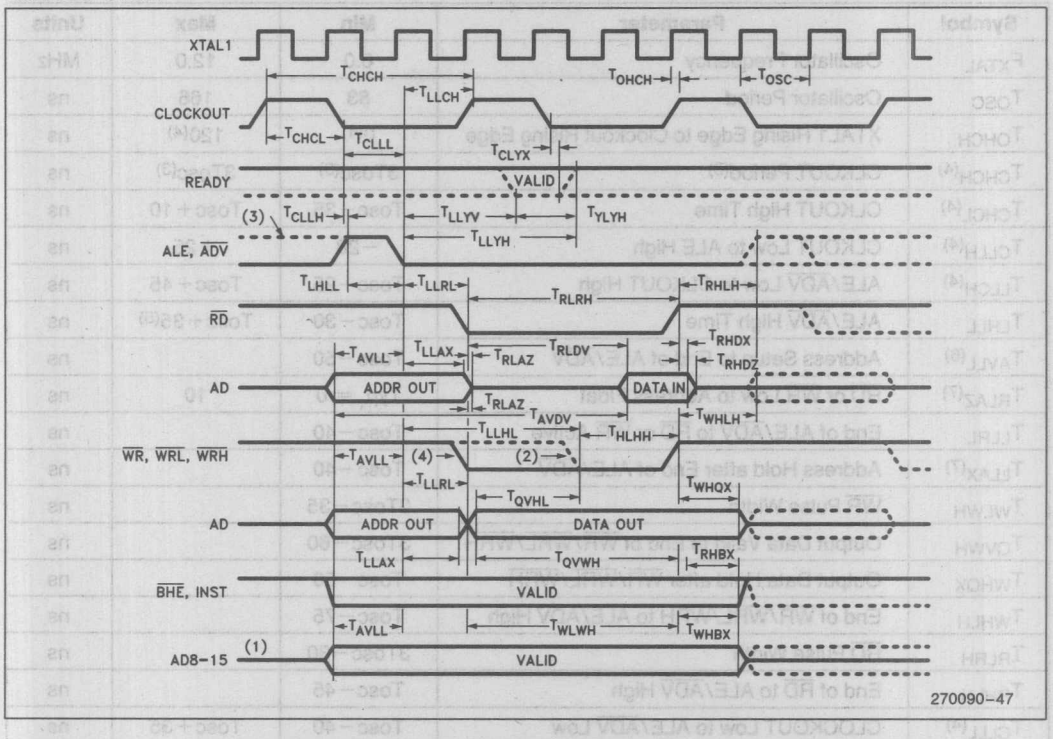
NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+55 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	12.0	MHz
T_{OSC}	Oscillator Period	83	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0(4)	120(4)	ns
T_{CHCH}(4)	CLKOUT Period(3)	3Tosc(3)	3Tosc(3)	ns
T_{CHCL}(4)	CLKOUT High Time	Tosc−35	Tosc+10	ns
T_{CLLH}(4)	CLKOUT Low to ALE High	−20	+25	ns
T_{LLCH}(4)	ALE/$\overline{ADV}$ Low to CLKOUT High	Tosc−25	Tosc+45	ns
T_{LHLL}	ALE/$\overline{ADV}$ High Time	Tosc−30	Tosc+35(5)	ns
T_{AVLL}(6)	Address Setup to End of ALE/$\overline{ADV}$	Tosc−50		ns
T_{RLAZ}(7)	$\overline{RD}$ or $\overline{WR}$ Low to Address Float	Typ. = 0	10	ns
T_{LLRL}	End of ALE/$\overline{ADV}$ to $\overline{RD}$ or $\overline{WR}$ Active	Tosc−40		ns
T_{LLAX}(7)	Address Hold after End of ALE/$\overline{ADV}$	Tosc−40		ns
T_{WLWH}	$\overline{WR}$ Pulse Width	3Tosc−35		ns
T_{QVWH}	Output Data Valid to End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	3Tosc−60		ns
T_{WHQX}	Output Data Hold after $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	Tosc−50		ns
T_{WHLH}	End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$ to ALE/$\overline{ADV}$ High	Tosc−75		ns
T_{RLRH}	$\overline{RD}$ Pulse Width	3Tosc−30		ns
T_{RHLH}	End of $\overline{RD}$ to ALE/$\overline{ADV}$ High	Tosc−45		ns
T_{CLLL}(4)	CLOCKOUT Low to ALE/$\overline{ADV}$ Low	Tosc−40	Tosc+35	ns
T_{RHBX}(4)	$\overline{RD}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc−25	Tosc+30	ns
T_{WHBX}(4)	$\overline{WR}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc−50	Tosc+100	ns
T_{HLHH}	$\overline{WRL}$, $\overline{WRH}$ Low to $\overline{WRL}$, $\overline{WRH}$ High	2Tosc−35	2Tosc+40	ns
T_{LLHL}	ALE/$\overline{ADV}$ Low to $\overline{WRL}$, $\overline{WRH}$ Low	2Tosc−30	2Tosc+55	ns
T_{QVHL}	Output Data Valid to $\overline{WRL}$, $\overline{WRH}$ Low	Tosc−60		ns

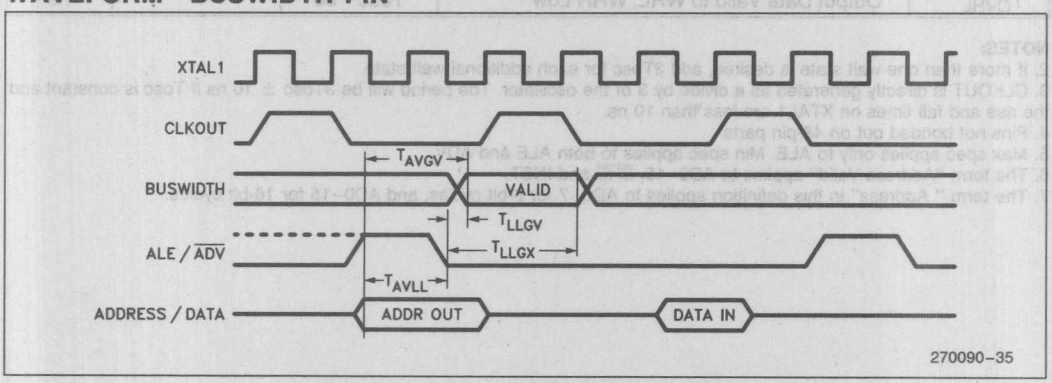
NOTES:
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{ADV}$.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.
7. The term " Address" in this definition applies to AD0–7 for 8-bit cycles, and AD0–15 for 16-bit cycles.

WAVEFORM



NOTES:
(1) 8-bit bus only.
(2) 8-bit or 16-bit bus and write strobe mode selected.
(3) When $\overline{ADV}$ selected.
(4) 8- or 16-bit bus and no write strobe mode selected.

WAVEFORM—BUSWIDTH PIN



A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

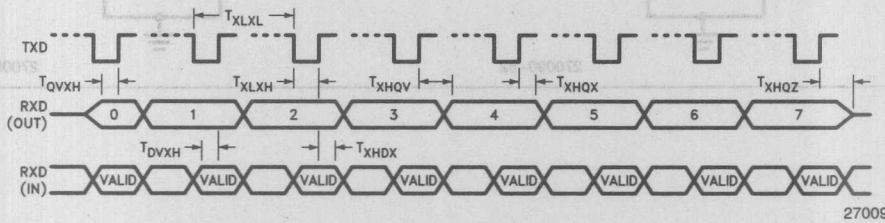
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 200$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$5T_{OSC}$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT WAVEFORM—SHIFT REGISTER MODE

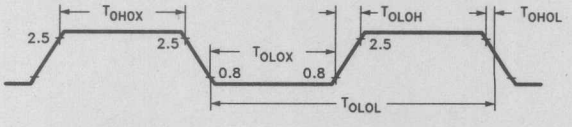


270090-36

EXTERNAL CLOCK DRIVE

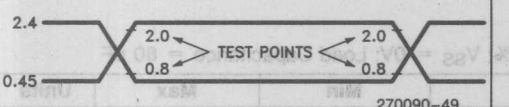
Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	12	MHz
T_{OHOX}	High Time	25		ns
T_{OLOX}	Low Time	25		ns
T_{OLOH}	Rise Time		15	ns
T_{OHOL}	Fall Time		15	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270090-48

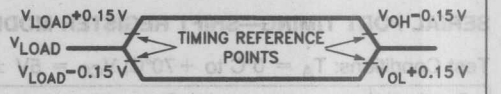
A.C. TESTING INPUT, OUTPUT WAVEFORM



270090-49

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

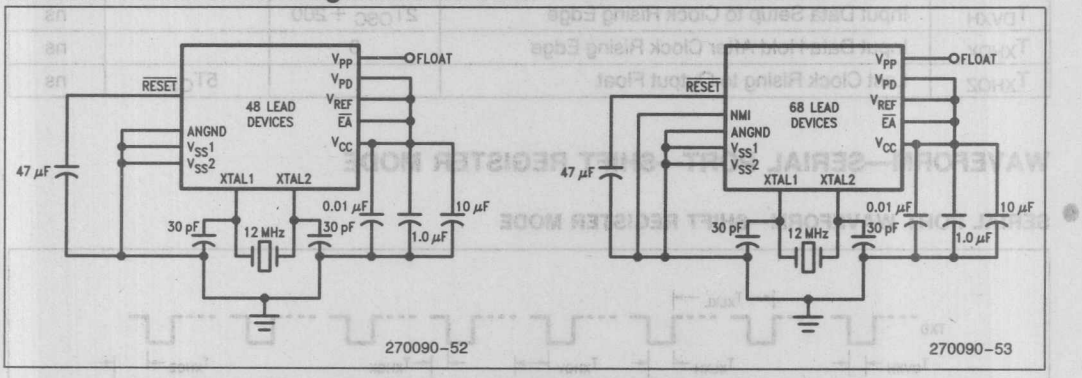
FLOAT WAVEFORM



270090-51

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

Minimum Hardware Configuration Circuits



270090-52

270090-53

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

OPERATING CONDITIONS

V_{CC}, V_{PD}, V_{REF} ..... 4.5V to 5.5V
V_{SS}, ANGND 0.0V
T_A ..... 0°C to 70°C
F_{OSC} ..... 6.0 to 12.0 MHz
Test Conditions:
V_{REF} 5.120V

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		1024	1024	Levels	
		10	10	Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	+2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients:					
Offset	0.009			LSB/°C	1
Full Scale	0.009			LSB/°C	1
Differential Non-Linearity	0.009			LSB/°C	1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Delay		$3T_{OSC}$ − 50	$3T_{OSC}$ + 50	ns	1, 3
Sample Time		$12T_{OSC}$ − 50	$12T_{OSC}$ + 50	ns	1
Sampling Capacitor			2	pF	

NOTES:
* These values are expected for most parts at 25°C.
** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12V, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal-based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the sample delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

EPROM CHARACTERISTICS

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 19 shows how the pins are renamed. Figure 20 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 18). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 18. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCR is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

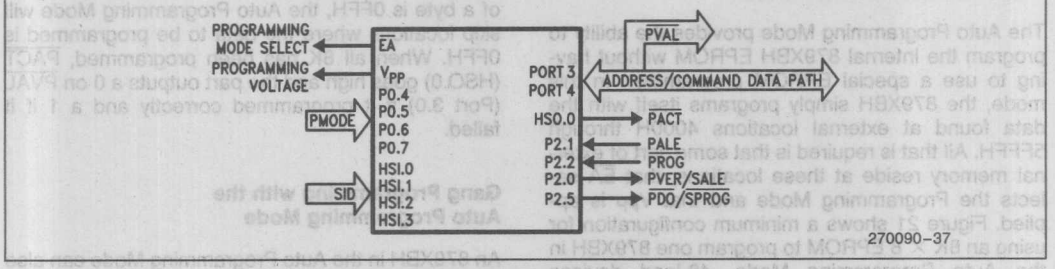


Figure 19. Programming Mode Pin Functions

Name	Function
PMODE	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	Programming ALE input. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming Pulse. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	Programming Active. Used in the Auto Programming Mode to indicate when programming activity is complete.
PVER	Program Verified. A signal output after a programming operation by parts in the Slave Programming Mode and after programming in the Auto Configuration Byte Programming Mode. This signal is on Port 2.0 and is asserted as a logic 1 if the bytes program correctly.
$\overline{\text{PVAL}}$	Program Valid. These signals indicate the success or failure of programming in the Auto Programming Mode and when using this mode for gang programming. For the Auto Programming Mode this signal is asserted at Port 3.0. When using this mode for gang programming, all bits of Port 3 and Port 4 are asserted to indicate programming validity of the various slaves. A zero indicates successful programming.
$\overline{\text{PDO}}$	Programming Duration Overflowed. A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	Slave ALE. Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	Slave Programming Pulse. Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indicates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Should have pullups to V_{CC} (15 KΩ).

Figure 20. Programming Mode Pin Definitions

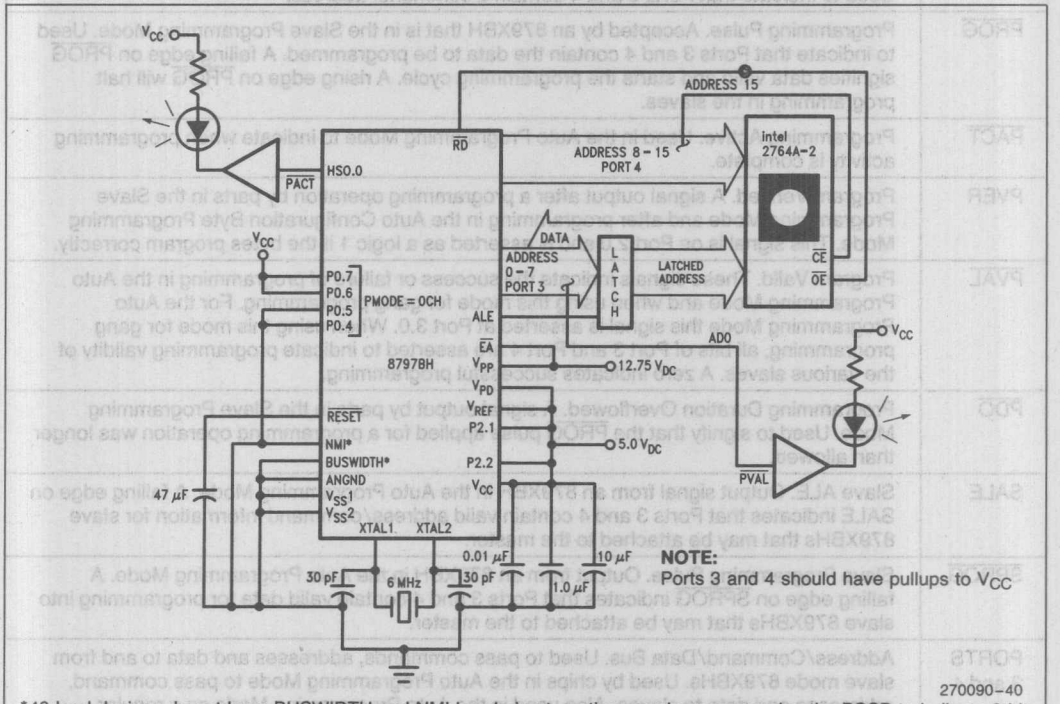
AUTO PROGRAMMING MODE

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that $\overline{EA}$ selects the Programming Mode and that V_{PP} is applied. Figure 21 shows a minimum configuration for using an 8K × 8 EPROM to program one 879XBH in the Auto Programming Mode. 48-lead devices (8795BH) must first use the Auto Configuration Byte Programming Mode to load the Programming Chip Configuration Byte (PCCB) for 8-bit Bus Cycles (PCCB = XXXXXX01B) in order to use the basic circuit presented in Figure 21 (see Auto Configuration Byte Programming Mode). 68-lead devices have Buswidth available externally to indicate the bus mode.

The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, $\overline{PACT}$ (HSO.0) goes high and the part outputs a 0 on $\overline{PVAL}$ (Port 3.0) if it programmed correctly and a 1 if it failed.

Gang Programming with the Auto Programming Mode

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave $\overline{PROG}$ ($\overline{SPROG}$) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 22 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

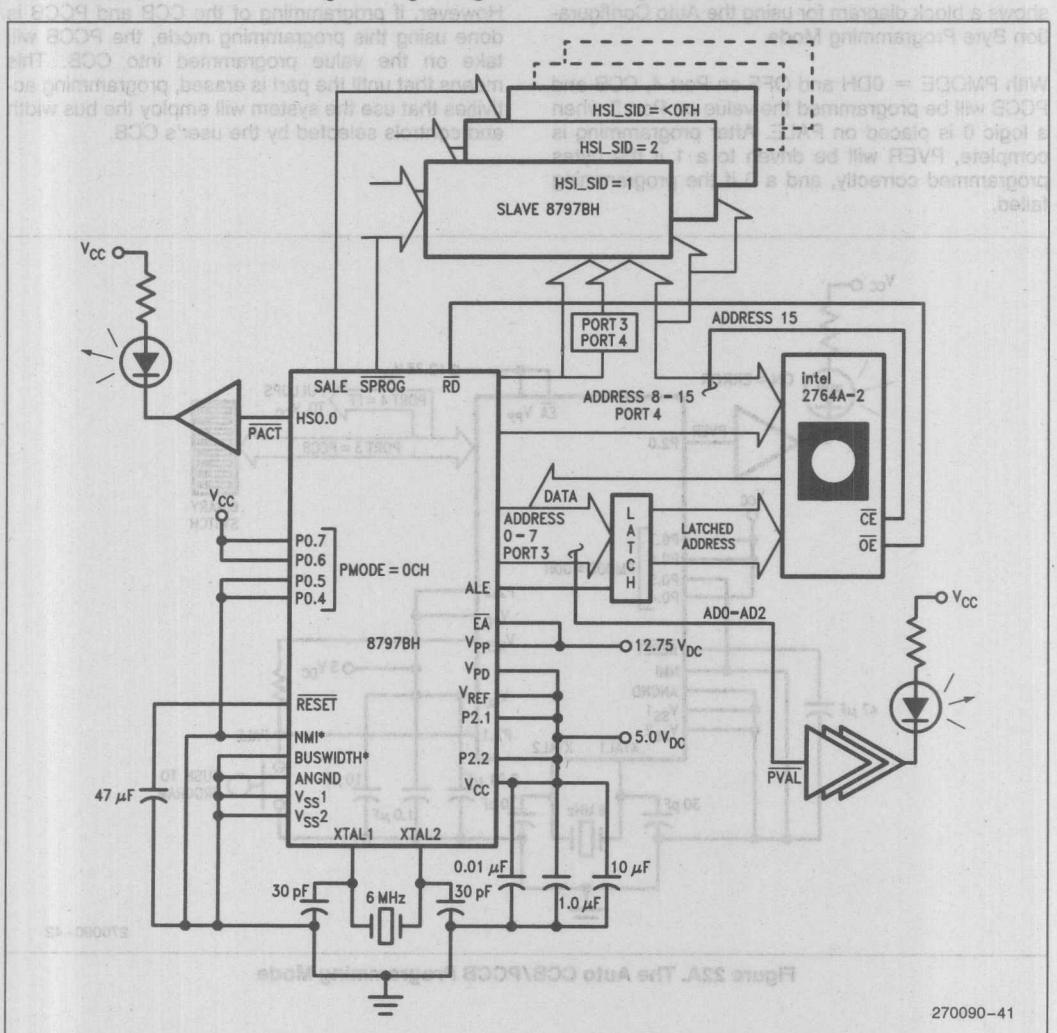


*48-lead devices do not have BUSWIDTH and NMI bonded out, so they require programming the PCCB to indicate 8-bit bus mode.

Figure 21. The Auto Programming Mode

The master 879XBH first reads a word from the external memory controlled by ALE, $\overline{RD}$ and $\overline{WR}$. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on $\overline{SPROG}$. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete, $\overline{PACT}$ goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.



NOTE:
$\overline{EA}$ and V_{PP} on slaves must be at $+12.75\ V_{DC}$. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to V_{CC}. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

*48-lead devices do not have NMI and BUSWIDTH bonded out, so they require programming the PCCB to indicate 8-bit bus mode to the master.

Figure 22. Gang Programming with the Auto Programming Mode

AUTO CONFIGURATION BYTE PROGRAMMING MODE

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 22A shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and OFF on Port 4, CCB and PCCB will be programmed the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on EA puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming of the CCB and PCCB is done using this programming mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system will employ the bus width and controls selected by the user's CCB.

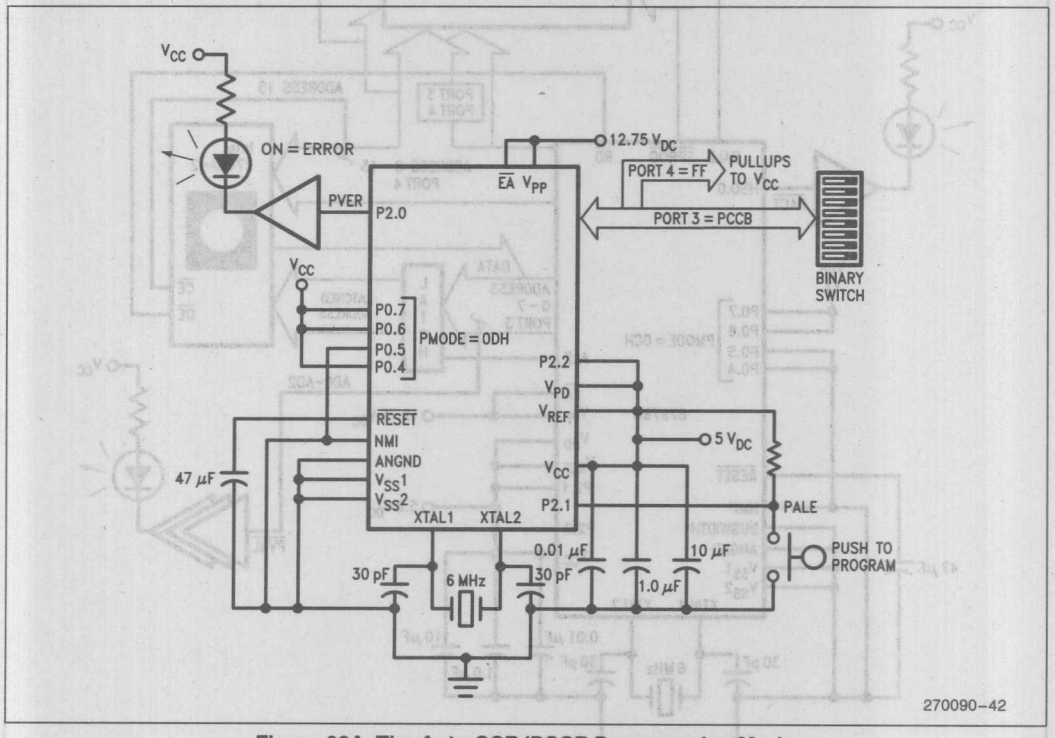


Figure 22A. The Auto CCB/PCCB Programming Mode

SLAVE PROGRAMMING MODE

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding PALE pulses immediately preceding is equally unpredictable.

Slave Programming Commands

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 4). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each Slave Programming Mode command.

Table 4. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 23 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

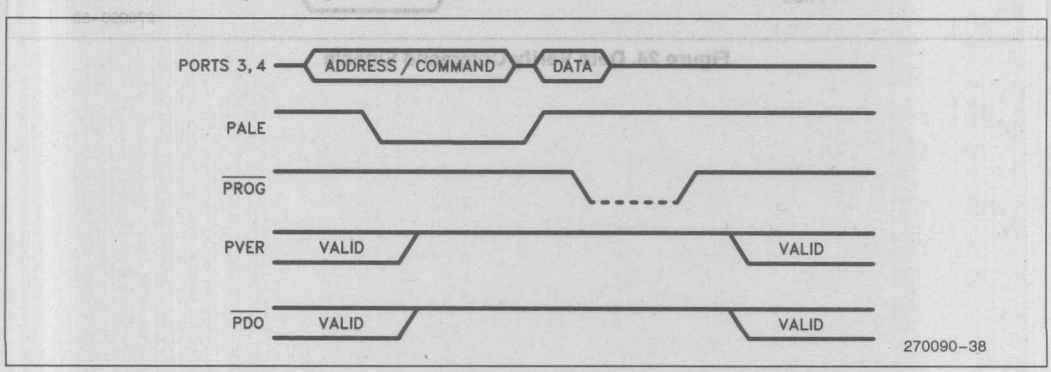


Figure 23. Data Program Signals in Slave Programming Mode

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 or 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{PROG}$ from the programmer governs when the slaves drive the bus. Figure 24 shows the relationship of Ports 3 and 4 to PALE and $\overline{PROG}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND — When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{PROG}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 24.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

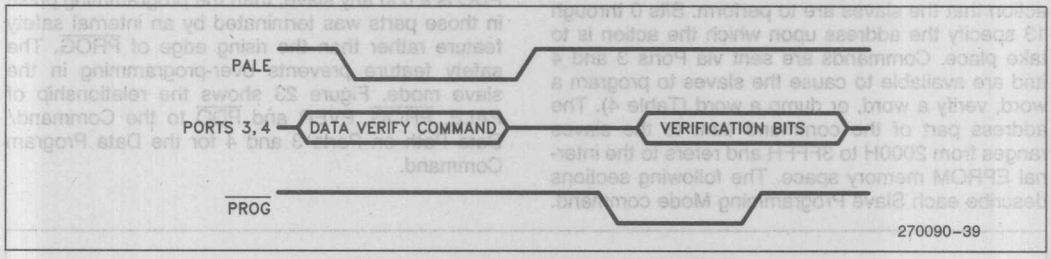


Figure 24. Data Verify Command Signals

Gang Programming with the Slave Programming Mode

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{PDO}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{PDO}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the auto programming mode could be the master programmer if 15 or fewer slaves need to be programmed (See Gang Programming with the Auto Programming Mode).

RUN-TIME PROGRAMMING

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to V_{PP}. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation - internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few microseconds. An inteligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming™ Algorithm be implemented.

After the programming of a location has started, care must be taken to insure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 26 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self." The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:
     POP   temp
     POP   address_temp          ;take parameters from the STACK
     POP   data--temp
     PUSH  temp

     PUSHF                       ;save current status
     LDB   int_mask ,#enable_swt_only  ;enable only swt interrupts

     LDB   HSO_COMMAND ,#SWT0_ovf      ;load swt command to interrupt
     ADD   HSO_TIME,TIMER1,#program_pulse  ;when program pulse time
                                 ;has elasped
     EI
     ST    data_temp, [address_temp]
     CALL  201AH
     POPF
     RET
SWT_ISR:
     . . .
swt0_expired:
     POP   0
     RET
     . . .
```

Figure 26. Programming the EPROM from Internal Memory Execution

ROM/EPROM PROGRAM LOCK

Protection mechanisms have been provided on the ROM and EPROM versions of the 809XBH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

PROGRAM LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable V_{PP} to the EPROM. This protects the entire EPROM 2000H-3FFFH from inadvertant or unauthorized programming.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H-3FFFH if the slave program counter is in the range 2000H-3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even through the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110b).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H-202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H-11FFH and 4000H-5FFFH. Unspecified data appears at the low addresses. Internal EPROM/ROM is dumped to 4000H-5FFFH beginning with internal address 2000H. When the ROM/EPROM dump is complete the CPU will enter a JUMP-ON-SELF condition.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:

Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

MODIFIED QUICK-PULSE PROGRAMMING™ ALGORITHM

The Modified Quick-Pulse Programming™ Algorithm calls for each EPROM location to receive 25 separate 100 μs (±5 μs) programming cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with V_{PP} = 12.75V ±0.25V and V_{CC} = 5.0V ±0.5V.

SIGNATURE WORD

The 879XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 5. 8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

POWER SUPPLY SEQUENCE WHILE PROGRAMMING

For any 879XBH that is in any programming mode, high voltages must be applied to the device. To avoid damaging the parts, the following rules must not be violated.

RULE #1— V_{PP} must not have a low impedance path to ground when V_{CC} is above 4.5V.

RULE #2— V_{CC} must be above 4.5V before V_{PP} can be higher than 5.0V.

RULE #3— V_{PP} must be within 1V of V_{CC} while V_{CC} is below 4.5V.

RULE #4— All voltages must be within tolerance and the oscillator stable before $\overline{RESET}$ rises.

RULE #5— $\overline{EA}$ must be brought high to place the part in programming mode before V_{PP} is brought high.

To adhere to these rules, the following power up and power down sequences can be followed.

POWER UP

$\overline{RESET}$ = 0;
CLOCK ON; if using an external clock
; instead of an oscillator
$V_{CC} = V_{PP} = V_{EA}$ = 5V;
PALE = $\overline{PROG}$ = PORT 34 = V_{IH};*
SID AND PMODE VALID;
$\overline{EA}$ = 12.75V;
V_{PP} = 12.75V;
WAIT; wait for supplies and clock to
; settle
$\overline{RESET}$ = 5V;
WAIT Tshll; See Data Sheet
BEGIN;

POWER DOWN

$\overline{RESET}$ = 0;
V_{PP} = 5V;
$\overline{EA}$ = 5V;
PALE = $\overline{PROG}$ = SID = PMODE = PORT34 = 0V;
$V_{CC} = V_{PP} = V_{EA}$ = 0V;
CLOCK OFF;

*V_{IH} = Logical "1", 2.4V Minimum

One final note on power up, power down. The maximum limit on V_{PP} must never be violated, even for an instant. Therefore, an RC rise to the desired V_{PP} is recommended. V_{PP} is also sensitive to instantaneous voltage steps. This also can be avoided by using an RC ramp on V_{PP}.

EPROM SPECIFICATIONS

A.C. EPROM PROGRAMMING CHARACTERISTICS

Operating Conditions: Load Capacitance = 150 pF, T_A = 25°C ±5°C, V_{CC}, V_{PD}, V_{REF} = 5.0V ± 0.5V, V_{SS}, AGND = 0V, V_{PP} = 12.75V ± 0.25V, $\overline{EA}$ = 11V ± 2.0V, f_{osc} = 6.0 MHz

Symbol	Parameter	Min	Max	Units
T_{AVLL}	ADDRESS/COMMAND Valid to PALE Low	0		T_{osc}
T_{LLAX}	ADDRESS/COMMAND Hold After PALE Low	80		T_{osc}
T_{DVPL}	Output Data Setup Before $\overline{PROG}$ Low	0		T_{osc}
T_{PLDX}	Data Hold After $\overline{PROG}$ Falling	80		T_{osc}
T_{LLLH}	PALE Pulse Width	180		T_{osc}
T_{PLPH}	$\overline{PROG}$ Pulse Width	250 T_{osc}	100 µS + 144 T_{osc}	
T_{LHPL}	PALE High to $\overline{PROG}$ Low	250		T_{osc}
T_{PHLL}	$\overline{PROG}$ High to Next PALE Low	600		T_{osc}
T_{PHDX}	Data Hold After $\overline{PROG}$ High	30		T_{osc}
T_{PHVV}	$\overline{PROG}$ High to PVER/$\overline{PDO}$ Valid	500		T_{osc}
T_{LLVH}	PALE Low to PVER/$\overline{PDO}$ High	100		T_{osc}
T_{PLDV}	$\overline{PROG}$ Low to VERIFICATION/DUMP Data Valid	100		T_{osc}
T_{SHLL}	RESET High to First PALE Low (not shown)	2000		T_{osc}

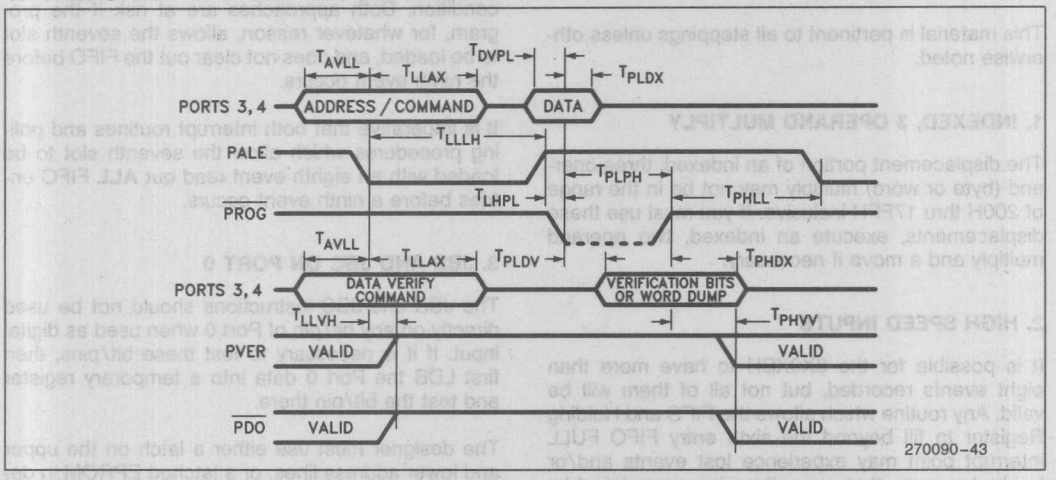
NOTE:
Run-time programming is done with F_{osc} = 6.0 MHz to 12.0 MHz, V_{CC}, V_{PD}, V_{REF} = 5V ± 0.5V, T_A = 25°C to ±5°C and V_{PP} = 12.75V ± 0.25V. For run-time programming over a full operating range, contact the factory. All windowed devices should be covered after programming.

D.C. EPROM PROGRAMMING CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
I_{PP}	V_{PP} Supply Current (Whenever Programming)		100	mA
V_{PP}	Programming Supply Voltage	12.75 ±0.25		V
V_{EA}	$\overline{EA}$ Programming Voltage	11 ±2.0		V

NOTE:
V_{PP} must be within 1V of V_{CC} while V_{CC} < 4.5V. V_{PP} must not have a low impedance path to ground or V_{SS} while V_{CC} > 4.5V.

WAVEFORM—EPROM PROGRAMMING



Reserved location warning: Intel Reserved addresses can not be used by applications which use 8X9XBH internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be 0FFFFH to ensure a harmless result. A memory map indicating reserved locations on the 8X9XBH is shown in Figure 27.

Intel Reserved locations, when mapped to external memory, must be filled with 0FFFFH to ensure compatibility with future parts.

	FFFFH
EXTERNAL MEMORY OR I/O	4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	2080H
RESERVED	2072H–207FH
SIGNATURE WORD	2070H–2071H
RESERVED	2030H–206FH
SECURITY KEY	2020H–202FH
RESERVED	201CH–201FH
SELF JUMP CODE (27H FEH)	201AH–201BH
RESERVED	2019H
CHIP CONFIGURATION BYTE	2018H
RESERVED	2012H-2017H
INTERRUPT VECTORS	2000H

Figure 27. Reserved Locations

8X9XBH ERRATA

This material is pertinent to all steppings unless otherwise noted.

1. INDEXED, 3 OPERAND MULTIPLY

The displacement portion of an indexed, three operand (byte or word) multiply may not be in the range of 200H thru 17FFH inclusive. If you must use these displacements, execute an indexed, two operand multiply and a move if necessary.

2. HIGH SPEED INPUTS

It is possible for the 8X9XBH to have more than eight events recorded, but not all of them will be valid. Any routine which allows the FIFO and Holding Register to fill beyond the sixth entry FIFO FULL interrupt point may experience lost events and/or trashed events that are otherwise unexplainable. The conditions are as follows:

If an eighth event is loaded into the seventh FIFO slot and the software does not unload the entire FIFO, the ninth event will be loaded into that eighth slot with a time tag that has no event status associated with it. Further events may or may not have event status bits set or you may lose events, depending on the level of the FIFO activity. The FIFO, under these conditions, must be completely emptied in order to return to normal functionality. What happens is that the event flip-flops are cleared just before the eighth slot is loaded.

Efficient interrupt routines which promptly utilize the sixth entry loaded (FIFO Full) interrupt vector to unload the FIFO, will never experience this condition because the entries in the FIFO are never more than six deep. In addition, polling procedures in a time domain which read out FIFO entries before the eighth event occurs will also not experience this condition. Both approaches are at risk if the program, for whatever reason, allows the seventh slot to be loaded, and does not clear out the FIFO before the ninth event occurs.

It is imperative that both interrupt routines and polling procedures which allow the seventh slot to be loaded with an eighth event read out **ALL** FIFO entries before a ninth event occurs.

3. JBS AND JBC ON PORT 0

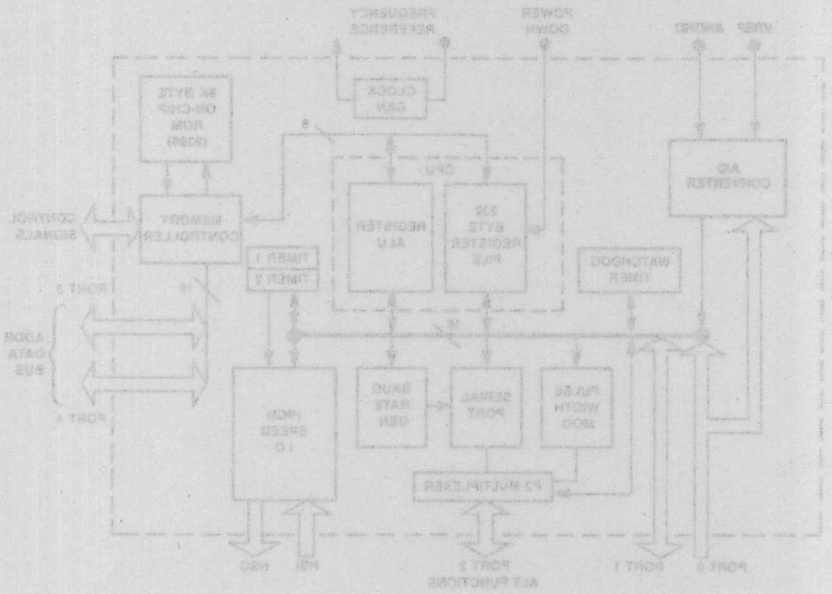
The JBS and JBC instructions should not be used directly on any bit/pin of Port 0 when used as digital input. If it is necessary to test these bit/pins, then first LDB the Port 0 data into a temporary register and test the bit/pin there.

The designer must use either a latch on the upper and lower address lines, or a latched EPROM in designs which will reset into an 8-bit memory system. The 48-pin devices (8095BH) did not bondout the BUSWIDTH pin in order to accommodate the package pin constraints of the DIP. As a result, the internal pullup on the BUSWIDTH input circuit pulls this input high inside these packages. A high condition on this input equates to a 16-bit bus mode externally, and during the CHIP CONFIGURATION BYTE fetch cycle, a logic design error in the port circuitry changes the CCB address about 25 ns after READ goes to a low condition. This invalidates the upper address byte on AD8–15. If there is no latch present on the upper address lines (and in an 8-bit only system, there typically would not be a latch), there is no guarantee that the correct data will be fetched for the Chip Configuration Register. Therefore, the CPU may never be able to realize that it is in an 8-bit system, unless a latch is on both Port 3 and Port 4.

DATA SHEET REVISION REVIEW

The following represents the key differences between this and the -003 version of the 8X9XBH Data Sheet. Please review this summary carefully.

1. The bus control figures and bus timing diagrams were modified to more accurately describe their operation. In particular the 8-bit bus modes now reflect the use of Write Strobe Mode.
2. Additional text was added to the Analog/Digital description of the conversion process to clarify its operation and usefulness.
3. Text was added to the interrupt description section to indicate the maximum transition speed of the input signal relative to the CPU's state timing. A figure was included to graphically demonstrate the interrupt response timing.
4. The pin descriptions were modified to indicate that V_{PP} must normally float in the application.
5. The input low voltage specification (V_{IL1}) was deleted and is covered by the V_{IL} specification.
6. A suggested minimum configuration circuit was added to the material.
7. The A/D Converter Specifications for Differential Non-Linearity has been corrected to be a maximum of +2 LSB's.
8. The EPROM programming section figures were corrected to indicate the correct interface to a 2764A-2. A reset circuit was added to these figures and the signal $\overline{PVAL}$ (Port 3.X and Port 4.X) is now identified as the valid signal for program verification in the Auto Programming Mode. Text was added to this section to reference the requirement of using the Auto Configuration Byte Programming Mode for 48-lead devices. Figure 22A was edited for corrections to the text, and now indicates PVER (Port 2.0). The EPROM circuits were corrected to show 6 MHz operation for programming devices from internal microcode.
9. The protected memory section was edited to indicate that the CPU will enter a "JUMP ON SELF" condition when ROM/EPROM dump mode is complete.
10. An 8X9XBH ERRATA section was added.
11. This REVISION REVIEW was added.



MCS®-96
809X-90, 839X-90

- 839X: an 809X with 8 Kbytes of On-Chip ROM
- High Speed Pulse I/O
- 10-Bit A/D Converter
- 6.25 μs 16 x 16 Multiply
- 6.25 μs 32/16 Divide
- 8 Interrupt Sources
- Pulse-Width Modulated Output
- 232 Byte Register File
- Memory-to-Memory Architecture
- Full Duplex Serial Port
- Five 8-Bit I/O Ports
- Watchdog Timer
- Four 16-Bit Software Timers

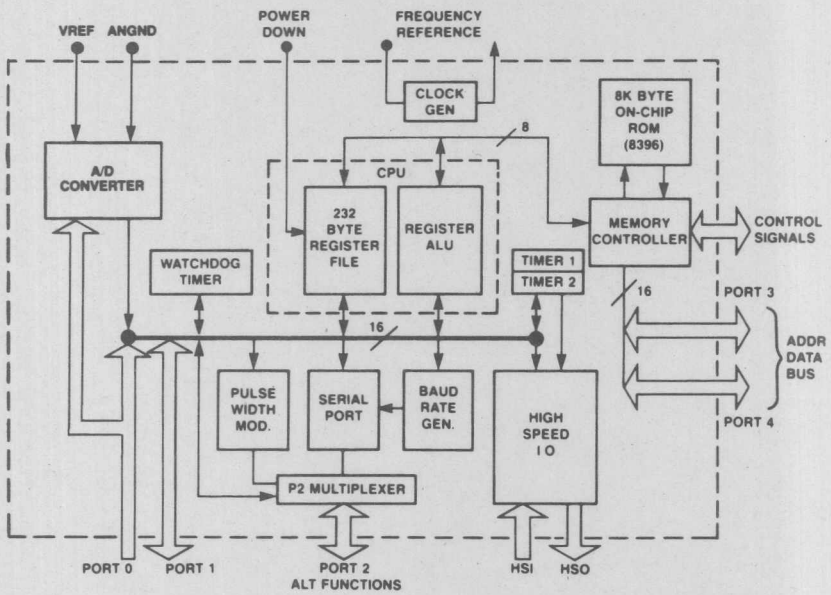
The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. Members with the "−90" suffix are described in this data sheet.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μs and a 16 x 16-bit multiply or 32/16-bit divide in 6.25 μs. Instruction execution times average 1 to 2 μs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at present times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095-90/8395-90 and 8097-90/8397-90.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.



270014-1

Figure 1. Block Diagram

October 1987
Order Number: 270014-004

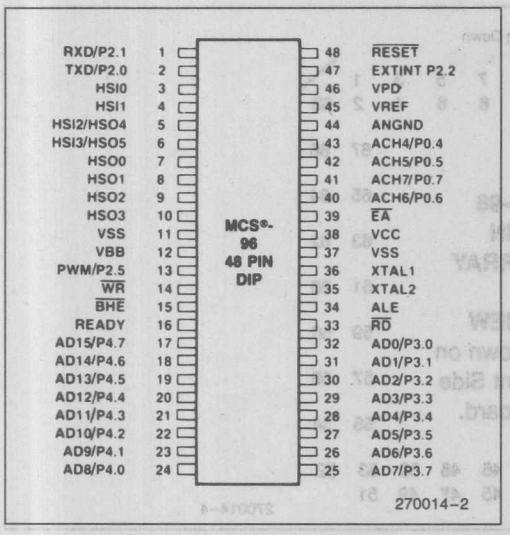


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

Options		68-Pin	48-Pin
Digital I/O	ROMLESS	8096-90	
	ROM	8396-90	
Analog and Digital I/O	ROMLESS	8097-90	8095-90
	ROM	8397-90	8395-90

Figures 2, 3 and 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin version comes in a Plastic Leaded Chip Carrier and a Pin Grid Array.

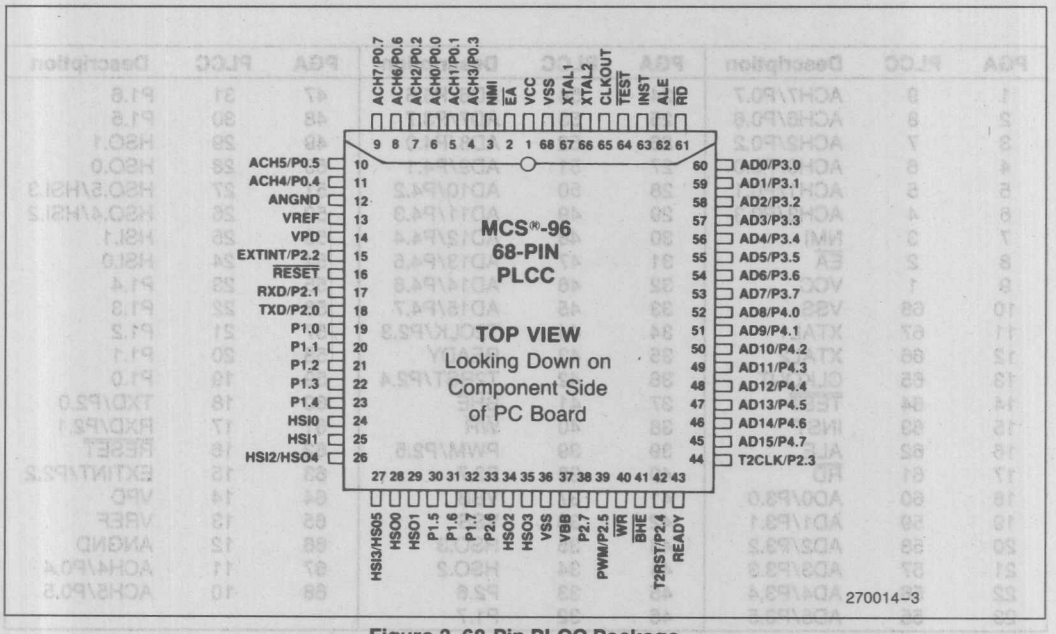


Figure 3. 68-Pin PLCC Package

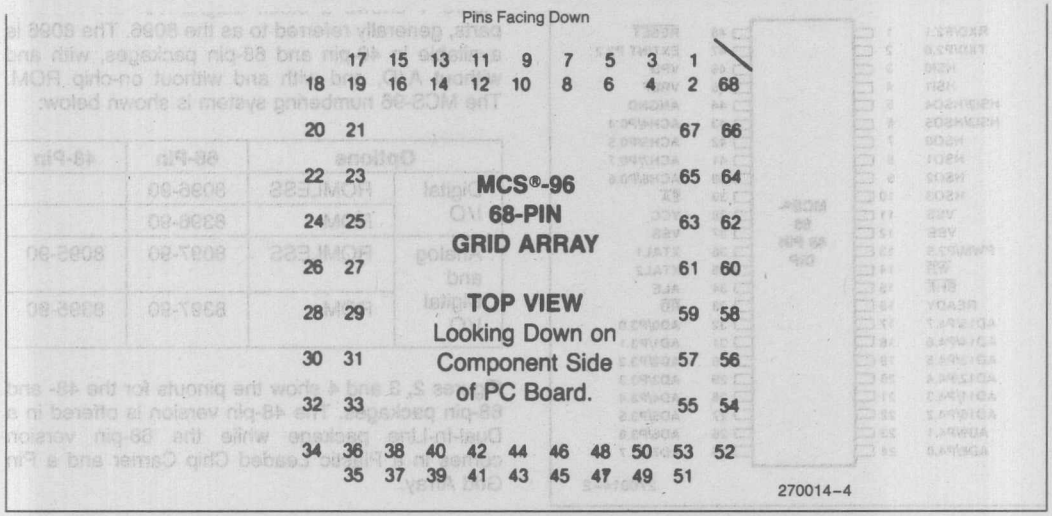


Figure 4. Pin Grid Array

PGA	PLCC	Description	PGA	PLCC	Description	PGA	PLCC	Description
1	9	ACH7/P0.7	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	$\overline{TEST}$	37	41	$\overline{BHE}$	60	18	TXD/P2.0
15	63	INST	38	40	$\overline{WR}$	61	17	RXD/P2.1
16	62	ALE	39	39	PWM/P2.5	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2
18	60	AD0/P3.0	41	37	VBB	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5
23	55	AD5/P3.5	46	32	P1.7			

FUNCTIONAL OVERVIEW

The following section is an overview of the 8096, the generic part number used to refer to the entire MCS-96 product family. Additional information is available in the Microcontroller Handbook, order number 210918-004.

CPU Architecture

The 8096 has 64 Kbyte addressability and uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems.)

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down if power is applied to the VPD pin.

Outside of the register file, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H—0017H	Register-mapped I/O (SFRs)
0018H—0019H	Stack Pointer
1FFEH—1FFFH	Ports 3 and 4
2000H—2011H	Interrupt Vectors
2012H—207FH	Factory Test Code
2080H	Reset Location

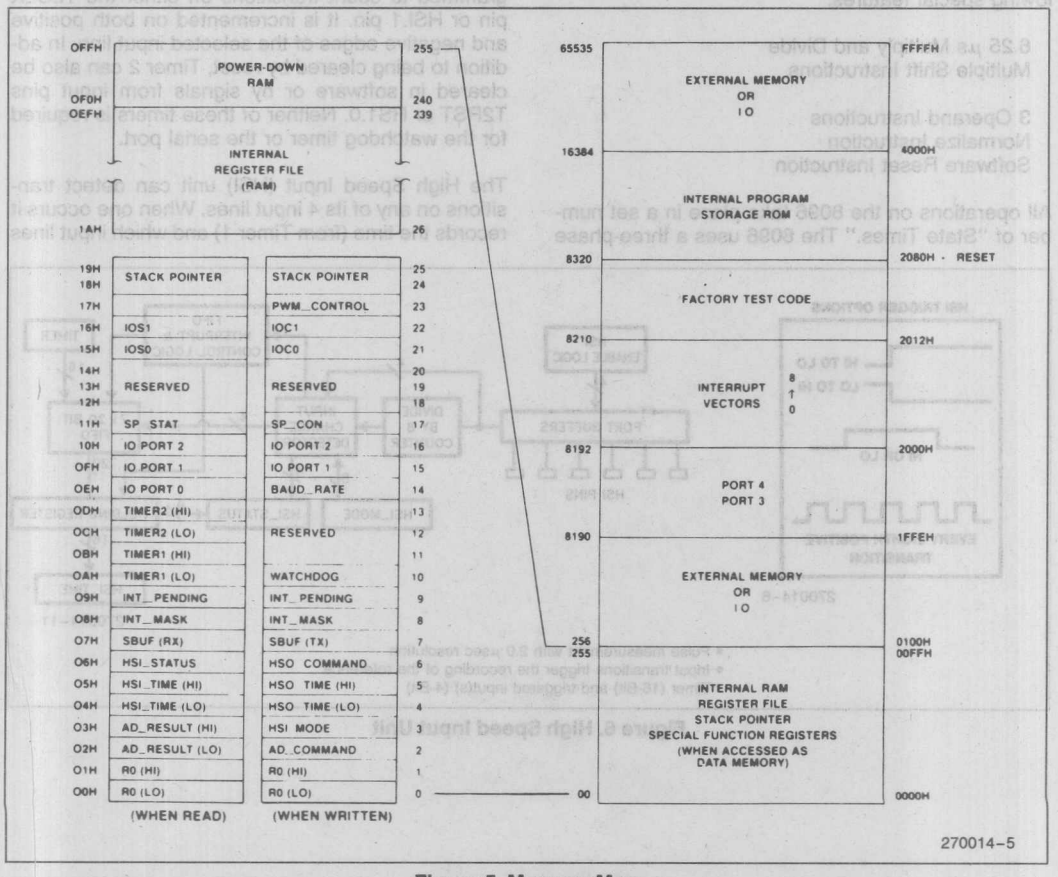


Figure 5. Memory Map

The 839x carries 8 Kbytes of on-chip ROM, occupying addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip ROM if the $\overline{EA}$ pin is externally held at a logical 1. If the $\overline{EA}$ pin is at a logical 0 these addresses access off-chip memory.

A memory map for the MCS-96 product family is shown in Figure 5.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096 is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

6.25 μs Multiply and Divide
Multiple Shift Instructions

3 Operand Instructions
Normalize Instruction
Software Reset Instruction

All operations on the 8096 take place in a set number of "State Times." The 8096 uses a three-phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 microseconds.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at preprogrammed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 microseconds, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HS1.0. Neither of these timers is required for the watchdog timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines

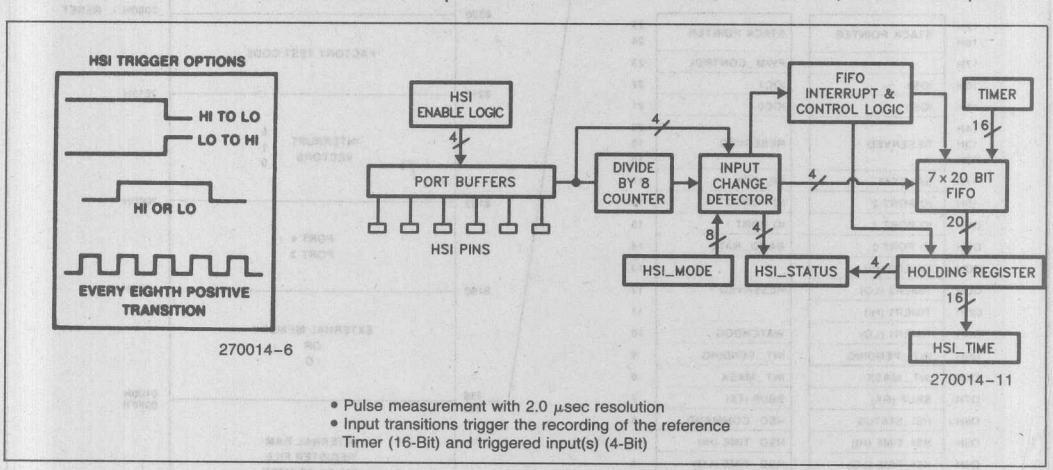


Figure 6. High Speed Input Unit

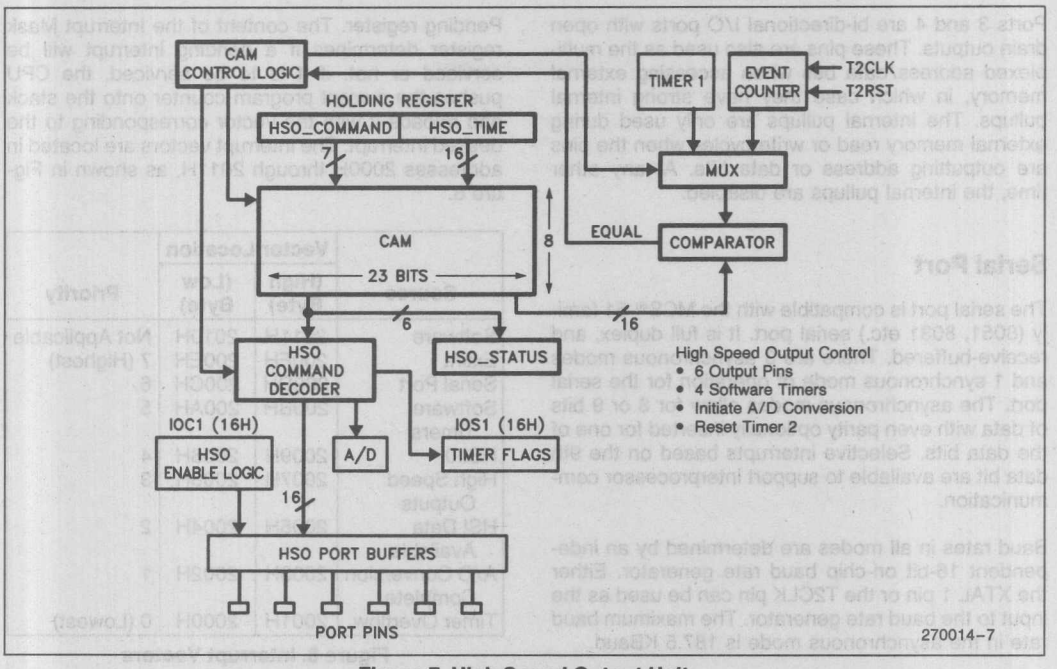


Figure 7. High Speed Output Unit

made the transition. This information is recorded with 2 microsecond resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 6. It can activate the HSI Data Available interrupt either when the Holding Registers is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

The High Speed Output (HSO) unit is shown in Figure 7. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timers flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096 in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D Converter.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is multi-functional port. Two of the pins are quasi-bidirectional while the remaining six are shared with other functions in the 8096, as shown below:

Port	Function	Alternate Function
P2.0	output	TXD (serial port transmit)
P2.1	input	RXD (serial port receive)
P2.2	input	EXTINT (external interrupt)
P2.3	input	T2CLK (Timer 2 clock)
P2.4	input	T2RST (Timer 2 reset)
P2.5	output	PWM (pulse-width modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled.

Serial Port

The serial port is compatible with the MCS®-51 family (8051, 8031 etc.) serial port. It is full duplex, and receive-buffered. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL 1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter

The analog-to-digital converter is a 10-bit, successive approximation converter. It has a fixed conversion time of 168 state times, (42 microseconds with a 12 MHz clock). The analog input must be in the range of 0 to VREF (normally, VREF = 5V). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

Interrupts

The 8096 has 20 interrupt sources which vector through 8 locations. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. if it is to be serviced, the CPU pushes the current program counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 8.

Source	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 8. Interrupt Vectors

At the end of the terminal routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the register file. This feature of the architecture provides for very fast context switching.

While the 8096 has a single priority level in the sense that any interrupt may be itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 8. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users' software.

Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. After 16 milliseconds, if not cleared, it will overflow, pulling down the $\overline{\text{RESET}}$ pin for two state times, causing the system to be reinitialized. This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$.

PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V).

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e., VCC drops to zero), if $\overline{\text{RESET}}$ is activated before VCC drops below spec and VPD continues to be held within spec, the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until VCC is within spec and the oscillator has stablized.

VREF

Reference voltage for the A/D converter (5V). VREF is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0 as digital input.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 μf capacitor (and not connected to anything else).

XTAL1

Input of the oscillator inverter and of the internal clock generator.

XTAL2

Output of the oscillator inverter.

CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.

$\overline{\text{RESET}}$

Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared and a jump to address 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.

TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI

A positive transition clears the watchdog timer, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.

INST

Output high during an external memory read indicates the read is an instruction fetch. INST needs to be latched on the falling edge of ALE.

$\overline{\text{EA}}$

Input for memory select (External Access). $\overline{\text{EA}}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven to 1. $\overline{\text{EA}}$ is not latched internally during RESET.

ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus, and is placed in a low condition during reset.

$\overline{\text{RD}}$

Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.

$\overline{\text{WR}}$

Write signal output to external memory. $\overline{\text{WR}}$ is activated only during external memory writes.

$\overline{BHE}$

Bus High Enable signal output to external memory. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bust. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or to both bytes (A0 = 0, $\overline{BHE}$ = 0). $\overline{BHE}$ is activated only when required during accesses to external memory. $\overline{BHE}$ can be ignored during read operations. This pin must be latched on the falling edge of ALE.

READY

The READY input is used to lengthen external memory bus cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high CPU operation continues in a normal manner. If the pin is low prior to the first rising edge of CLKOUT after ALE, the Memory Controller goes into a wait mode until the next negative transition in CLKOUT after ALE occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory bus is not being used, READY has no effect. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.

HSI

Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit.

HSO

Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.

Port 0

8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter.

Port 1

8-bit quasi-bidirectional I/O port.

Port 2

8-bit multi-functional port. Six of its pins are shared with other functions in the 8096, the remaining 2 are quasi-bidirectional.

Ports 3 and 4

8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups.

INSTRUCTION SET

The 8096 instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096 contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field in an indexed instruction contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	—	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	—	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↕	↕	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↕	?	↕	↕	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↕	?	↕	0	—	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↕	↕	↕	0	—	↕	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	↕	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT*					INDEXED*				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270014-9

NOTES:

* Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect+ or Long Indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

1. Number of state times shown for internal/external operands.
2. The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
3. State times shown for 16-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270014-10

NOTES:
1. Number of state times shown for internal/external operands.
3. The assembler does not accept this mnemonic.
4. The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
5. State times for stack located internal/external.

Conditional Jumps

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

Jump on Bit Clear or Bit Set

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	Bit Number							
Mnemonic	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ	OPCODE E0;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)

Single Register Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

Shift Instructions

Instr Mnemonic	Word		Instr Mnemonic	Byte		Instr Mnemonic	DBL WD		State Times
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

Special Control Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST (6)	FF	1	166	SKIP	00	2	4

Normalize

Mnemonic	Opcode	Bytes	State Times
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

6. This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
7. Execution will take at least 8 states, even for 0 shift.

FUNCTIONAL DEVIATIONS

Functional deviations from the 809x and 839x on the 809x-90 and 839x-90.

CPU Section

1. Indexed, 3 Operand Multiply—The displacement portion of an indexed, three word multiply may not be in the range of 200H thru 17FFH inclusive. This also applies to byte multiples that use 3 operands.
2. Add or Subtract with carry—The zero flag is both set and cleared by these instructions. Zero checking must be done after each operation.
3. EXT—This instruction never sets the N flag, and always sets the Z flag. The EXTB works correctly. Check the flags before executing an EXT instruction. Additionally, having more than two wait states during an EXT (extend word only) instruction may cause the instruction to give an incorrect result.
4. Read-Modify-Write on Interrupt Pending—A read-modify-write instruction on the interrupt pending register may cause interrupts that occur during execution of the instruction to be missed.
5. READY line—The READY line should not be brought low during the execution of an instruction that accesses HSI_TIME, SP_STAT or IOS1. It should also not be brought low for a data write during the instruction immediately preceding one of the above operations. Do not use wait states for program memory that holds these instructions. Also place a NOP between writes to slow memory and accesses to HSO_TIME, SP_STAT or IOS1.

   The READY line also should not be brought low for more than two state times when using the EXT (extend word) instruction.
6. Signed Divide—The V and VT flags may indicate an overflow after a signed divide when no overflow has occurred.
7. The sticky flag is not affected when a shift by zero is executed on an 8X9X-90.
8. The JBS and JBC instructions should not be used directly on Port 2.1 or any pins of Port 0 if used as digital input. If it is necessary to test these pins, first LD the port data into a temporary register, and then test the bit there.

HSI/HSO Section

1. HSI Timing—An event occurring within 16 state times of a prior event on the same HSI line may not be recorded. Additionally, an event occurring within 16 state times of a prior event on another HSI line may be recorded with a time tag one count earlier than expected. Events are defined as the condition the line is set to trigger on. The effective resolution is increased to 4 μs for such closely spaced events.
2. HSI Divide by 8 Mode—If an event on a pin set to look for every eighth transition occurs less than 16 state times after an event on any other pin, then the divide by 8 event will be recorded twice in the HSI FIFO. The time tag of the duplicate FIFO entry will be equal to that of the initial entry plus one. The programmer's software should detect and discard the second entry.
3. HSO Interrupts—Software timer interrupts cannot be generated by the HSO commands that reset Timer 2 or start an A to D conversion.
4. The first few instructions of an interrupt service routine should check IOS1.7 and exit if the Holding Register is not loaded. This will successfully clear unwanted events.

Serial Port Section

1. Serial Port Flags—Reading SP_STAT may not clear the TI or RI flag if that flag was set within two state times prior to the read. In addition, the parity error bit (RPE/RB8) may not be correct if it is read within two state times after RI is set.

Use the following code to replace ORB sp_image, SP_STAT.

```
SP_READ:
  LDB TEMP, SP_STAT
  ORB SP_IMAGE, SP_STAT
  JBS TEMP,5,SP_READ ; if TI bit is set
                     ; then read again
  JBS TEMP,6,SP_READ ; if RI bit is set
                     ; then read again
  ANDB SP_IMAGE,#7FH ; clear false
                     ; RB8/RPE
  ORB SP_IMAGE,TEMP  ; load correct
                     ; RB8/RPE
```

2. Serial Port Mode 0—The serial port is not tested in mode 0. The receive function in this mode does not work correctly. The receive function will not work unless the first bit shifted in is a one.

3. Serial Port Baud Value—Loading the baud rate register with 8000H (maximum baud rate, internal clock) may cause an 11 millisecond delay (at Fosc = 12 MHz) before the port is properly initialized. After initialization the port works properly. Include a 44000 state time delay after writing 8000H to the Baud Rate Register.

Standard I/O Section

1. Ports 3 and 4 (Internal Execution Mode Only)—To be used as outputs, Ports 3 and 4 each must be addressed as words but written to as bytes. To write to Port 3 use "ST temp, 1ffeh", where the low byte of "temp" contains the data for the port. To write to Port 4, use the DCB operator to generate the opcode sequence "0C3H, 001H, 0FFH, 01FH, (temp)", where the high byte of "temp" contains the data for the port. Ports 3 and 4 will not work as input ports.

   Also, when writing to Ports 3 and 4, the address of the port, (1FFEH, 1FFFH) will appear on the bus pins for 2 oscillator periods before the new data is presented to the pins. Since normal bus control signals (ALE, RD, etc.) are suppressed during writes to these addresses, there is no way to latch the data and prevent this address "glitch" to the outside world. If this presents a problem in an application, port reconstruction must be done at another address as described in the MCS-96 Hardware Design Information Chapter.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to +70°C
Storage Temperature . . . . . −40°C to +150°C
Voltage from Any Pin to
V_{SS} or ANGND . . . . . −0.3V to +7.0V
Average Output Current from Any Pin . . . . . 10 mA
Power Dissipation . . . . . 1.5 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.4	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	VPD Supply Current		1	mA	Normal operation and Power-Down
I_{REF}	V_{REF} Supply Current		8	mA	
I_{LI}	Input Leakage Current to all pins of HSI, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	Input Leakage to Port 0		±3	μA	V_{IN} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to all pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	0.3	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and $\overline{RESET}$ and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120V.

Resolution ..... ±0.001 VREF
Accuracy ..... ±0.004 VREF
Differential nonlinearity ..... ±0.002 VREF max
Integral nonlinearity ..... ±0.004 VREF max
Channel-to-channel matching ..... ±1 LSB
Crosstalk (DC to 100 KHz) ..... −60 dB max

A.C. CHARACTERISTICS

(VCC, VPD = 4.5 to 5.5 Volts; T_A = 0°C to 70°C; fosc = 6.0 to 12.0 MHz)

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

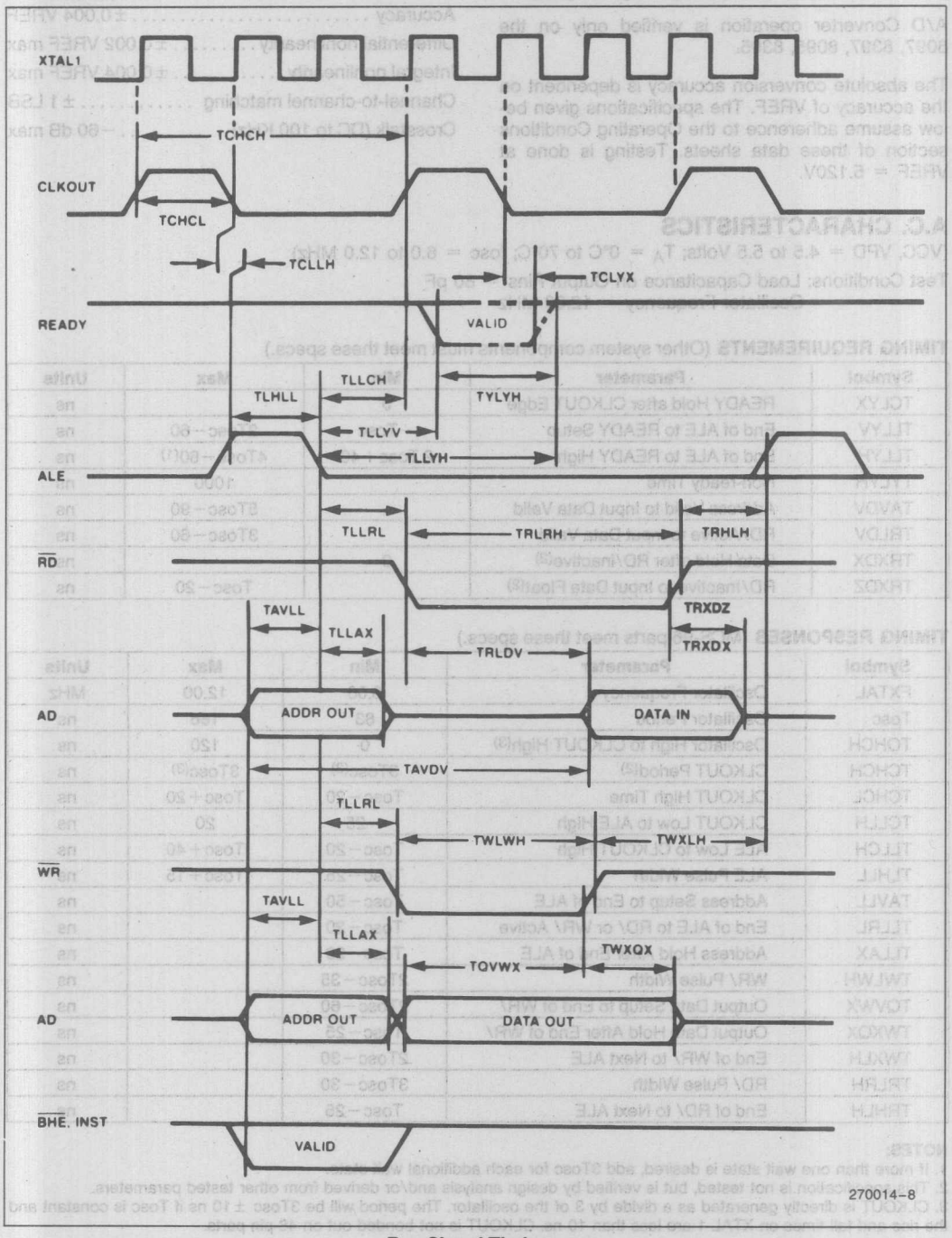
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Edge	0		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc−60	ns
TLLYH	End of ALE to READY High	2 Tosc+40	4Tosc−60(1)	ns
TYLYH	Non-ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc−90	ns
TRLDV	RD/Active to Input Data Valid		3Tosc−60	ns
TRXDX	Data Hold after RD/inactive(2)	0		ns
TRXDZ	RD/Inactive to Input Data Float(2)		Tosc−20	ns

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TOHCH	Oscillator High to CLKOUT High(3)	0	120	ns
TCHCH	CLKOUT Period(2)	3Tosc(3)	3Tosc(3)	ns
TCHCL	CLKOUT High Time	Tosc−20	Tosc+20	ns
TCLLH	CLKOUT Low to ALE High	−25	20	ns
TLLCH	ALE Low to CLKOUT High	Tosc−20	Tosc+40	ns
TLHLL	ALE Pulse Width	Tosc−25	Tosc+15	ns
TAVLL	Address Setup to End of ALE	Tosc−50		ns
TLLRL	End of ALE to RD/ or WR/ Active	Tosc−20		ns
TLLAX	Address Hold After End of ALE	Tosc−20		ns
TWLWH	WR/ Pulse Width	2Tosc−35		ns
TQVWX	Output Data Setup to End of WR/	2Tosc−60		ns
TWXQX	Output Data Hold After End of WR/	Tosc−25		ns
TWXLH	End of WR/ to Next ALE	2Tosc−30		ns
TRLRH	RD/ Pulse Width	3Tosc−30		ns
TRHLH	End of RD/ to Next ALE	Tosc−25		ns

NOTES:
1. If more than one wait state is desired, add 3Tosc for each additional wait state.
2. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns. CLKOUT is not bonded out on 48-pin parts.

WAVEFORM



Bus Signal Timings

MCS®-96
809XBH/839XBH/879XBH
Express

- **Extended Temperature Range (−40°C to +85°C)**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

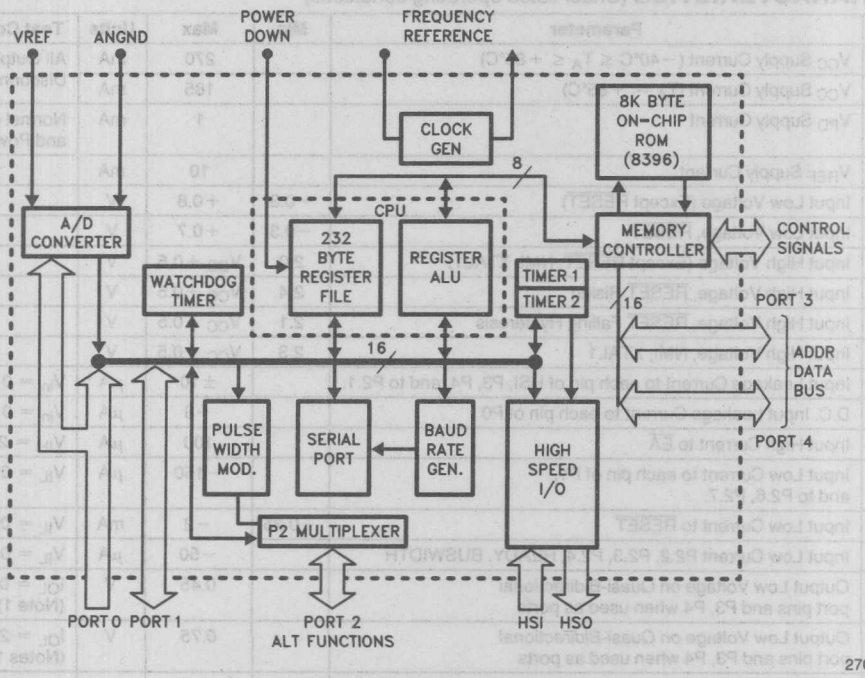
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to +70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.5V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



MCS®-96 Block Diagram

November 1987
Order Number: 270433-001

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . −40°C to +85°C

Storage Temperature ..... −40°C to +150°C

Voltage from V_{PP} or $\overline{EA}$ to V_{SS} or ANGND ..... −0.3V to +13.0V

Voltage from Any Other Pin to V_{SS} or ANGND ..... −0.3V to +7.0V*

Average Output Current from Any Pin .....10 mA

Power Dissipation.....1.5W

*This includes V_{PP} on ROM and CPU devices.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	−40	+85	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Under listed operating conditions)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current (−40°C ≤ T_A ≤ +85°C)		270	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current (T_A = +85°C)		185	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		10	mA	
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling Hysteresis	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.3	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+3	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−150	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	−0.25	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY, BUSWIDTH		−50	μA	V_{IL} = 0.45V
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.45	V	I_{OL} = 0.8 mA (Note 1)
V_{OL1}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.75	V	I_{OL} = 2.0 mA (Notes 1, 2, 3)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{RESET}$ and Bus/Control Pins		0.45	V	I_{OL} = 2.0 mA (Notes 1, 2, 3, 4)

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	$I_{OH} = -20\ \mu A$ (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	$I_{OH} = -200\ \mu A$ (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		μA	$V_{OH} = 2.4V$
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include TXD, RXD (Mode 0 only), PWM, and HSO pins. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins and Ports 3 and 4 when used as ports: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3. During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 8.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 15 mA.
   Total I_{OL} on Port 3 must not exceed 10 mA.
   Total I_{OL} on P2.5, P2.7, and Port 4 must not exceed 20 mA.
4. I_{OL} on HSO.X (X = 0, 4, 5) = 1.6 mA @ 0.5V.

A.C. CHARACTERISTICS (Under listed operating conditions)

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 10 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−70	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−80	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−120	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−100	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−25	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −125	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +40		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −75	ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+55 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

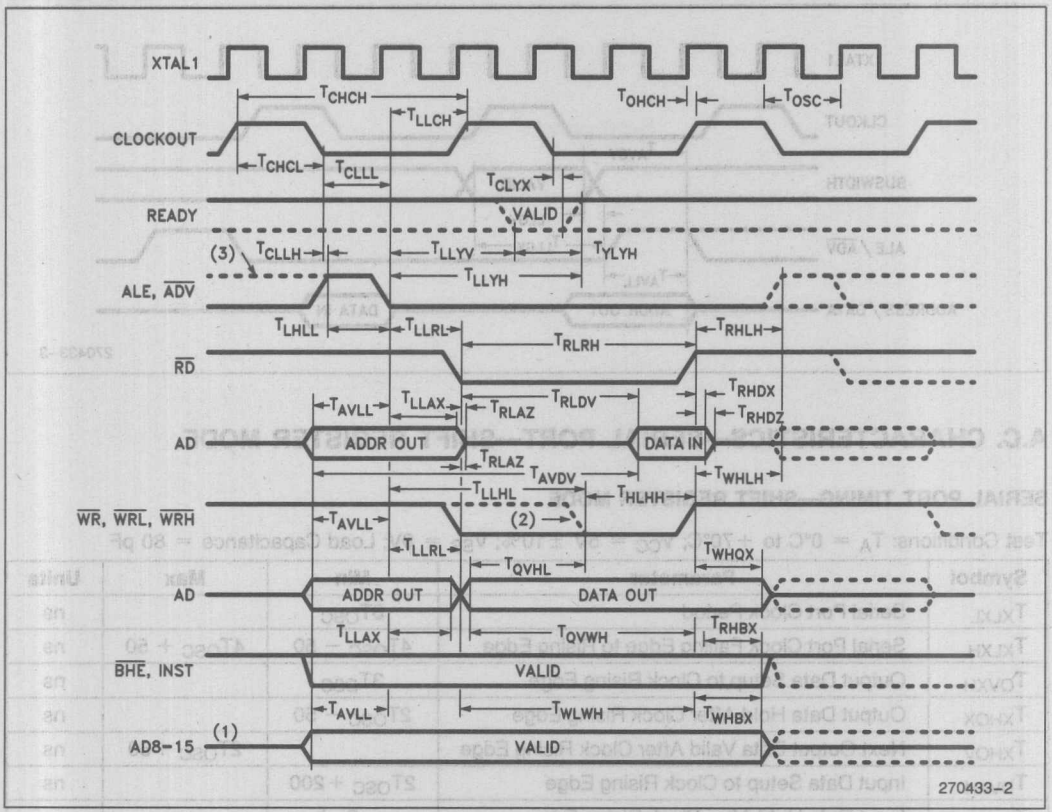
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	12.0	MHz
T_{OSC}	Oscillator Period	83	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0(4)	120(4)	ns
T_{CHCH}(4)	CLKOUT Period(3)	3Tosc(3)	3Tosc(3)	ns
T_{CHCL}(4)	CLKOUT High Time	Tosc−35	Tosc+10	ns
T_{CLLH}(4)	CLKOUT Low to ALE High	−20	+25	ns
T_{LLCH}(4)	ALE/$\overline{ADV}$ Low to CLKOUT High	Tosc−25	Tosc+45	ns
T_{LHLL}	ALE/$\overline{ADV}$ High Time	Tosc−30	Tosc+35(5)	ns
T_{AVLL}(6)	Address Setup to End of ALE/$\overline{ADV}$	Tosc−50		ns
T_{RLAZ}(7)	$\overline{RD}$ or $\overline{WR}$ Low to Address Float		25	ns
T_{LLRL}	End of ALE/$\overline{ADV}$ to $\overline{RD}$ or $\overline{WR}$ Active	Tosc−40		ns
T_{LLAX}(7)	Address Hold after End of ALE/$\overline{ADV}$	Tosc−40		ns
T_{WLWH}	$\overline{WR}$ Pulse Width	3Tosc−35		ns
T_{QVWH}	Output Data Valid to End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	3Tosc−60		ns
T_{WHQX}	Output Data Hold after $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	Tosc−50		ns
T_{WHLH}	End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$ to ALE/$\overline{ADV}$ High	Tosc−75		ns
T_{RLRH}	$\overline{RD}$ Pulse Width	3Tosc−30		ns
T_{RHLH}	End of $\overline{RD}$ to ALE/$\overline{ADV}$ High	Tosc−45		ns
T_{CLLL}(4)	CLOCKOUT Low to ALE/$\overline{ADV}$ Low	Tosc−40	Tosc+35	ns
T_{RHBX}(4)	$\overline{RD}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc−25	Tosc+30	ns
T_{WHBX}(4)	$\overline{WR}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc−50	Tosc+100	ns
T_{HLHH}	$\overline{WRL}$, $\overline{WRH}$ Low to $\overline{WRL}$, $\overline{WRH}$ High	2Tosc−35	2Tosc+40	ns
T_{LLHL}	ALE/$\overline{ADV}$ Low to $\overline{WRL}$, $\overline{WRH}$ Low	2Tosc−30	2Tosc+55	ns
T_{QVHL}	Output Data Valid to $\overline{WRL}$, $\overline{WRH}$ Low	Tosc−60		ns

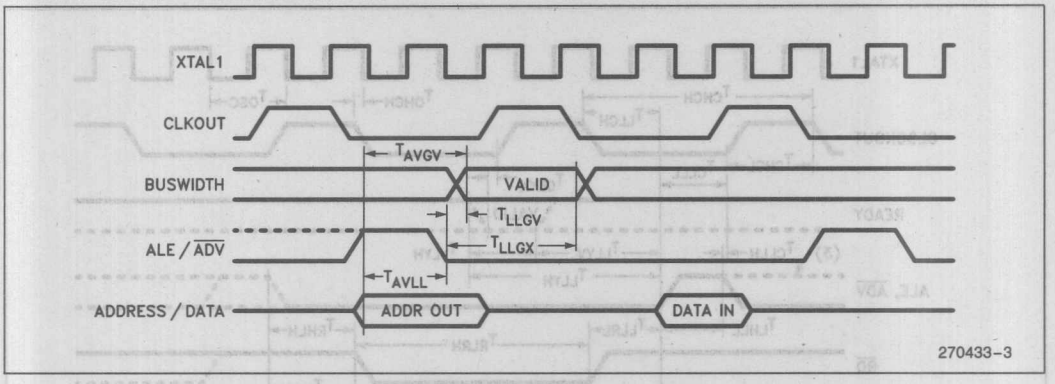
NOTES:
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{ADV}$.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.
7. The term " Address" in this definition applies to AD0–7 for 8-bit cycles, and AD0–15 for 16-bit cycles.

WAVEFORM



NOTES:
(1) 8-bit bus only.
(2) 8-bit bus; or when write strobe mode selected.
(3) When $\overline{ADV}$ selected.

WAVEFORM—BUSWIDTH PIN



A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

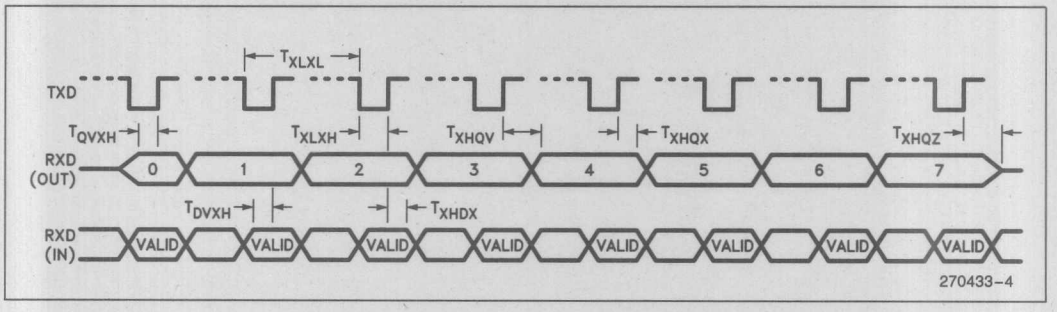
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 200$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$5T_{OSC}$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

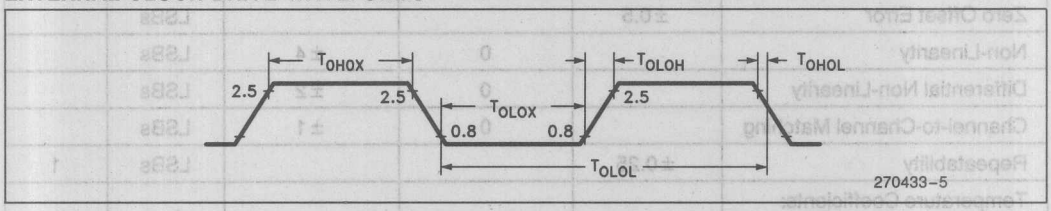
SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



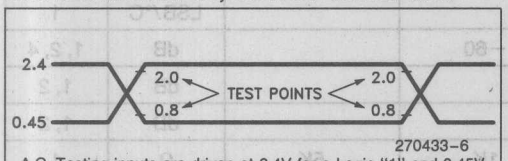
EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	12	MHz
T_{OHOX}	High Time	25		ns
T_{OLOX}	Low Time	25		ns
T_{OLOH}	Rise Time		15	ns
T_{OHOL}	Fall Time		15	ns

EXTERNAL CLOCK DRIVE WAVEFORMS

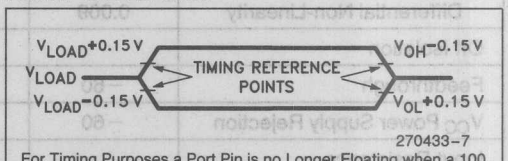


A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		1024 10	1024 10	Levels Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients:					
Offset	0.009			LSB/°C	1
Full Scale	0.009			LSB/°C	1
Differential Non-Linearity	0.009			LSB/°C	1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Delay		$3T_{OSC} - 50$	$3T_{OSC} + 50$	ns	1, 3
Sample Time		$12T_{OSC} - 50$	$12T_{OSC} + 50$	ns	1
Sampling Capacitor			2	pF	

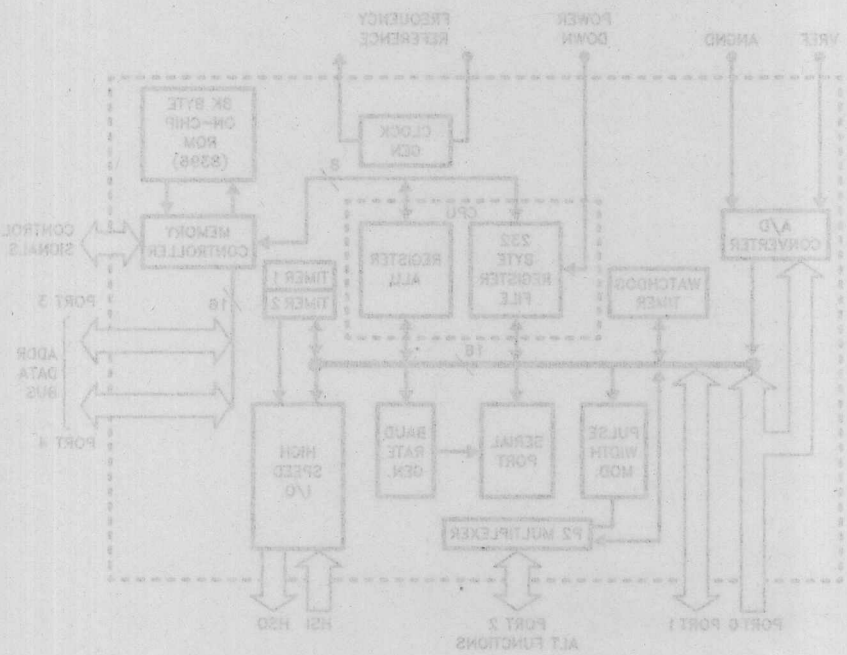
NOTES:
* These values are expected for most parts at 25°C.
** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

8096 BH Products

Code Memory	A/D	Analog Inputs	I/O Pins	Leads	Product	Package*
ROMless	No	0	48	68	8096BH	N
	Yes	4	32	48	8095BH	P LP
		8	48	68	8097BH	A LA N LN
ROM	No	0	48	68	8396BH	A LA TA N LN TN
	Yes	4	32	48	8395BH	P LP TP
		8	48	68	8397BH	A LA TA N LN TN
EPROM	Yes	4	32	48	8795BH	C LC
		8	48	68	8797BH	A LA R LR

Table 1. MCS®-96 Prefix Identification

*A = Commercial/No Burn-In 68L Ceramic F6A
N = Commercial/No Burn-In 68L PLCC
C = Commercial/No Burn-In 48L DIP (Ceramic)
P = Commercial/No Burn-In 48L DIP (Plastic)
TX = Extended Temp/No Burn-In
QX = Commercial/With Burn-In
LX = Extended Temp/With Burn-In



MCS®-96
809X-90, 839X-90

Express

- **Extended Temperature Range (−40°C to +85°C)**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

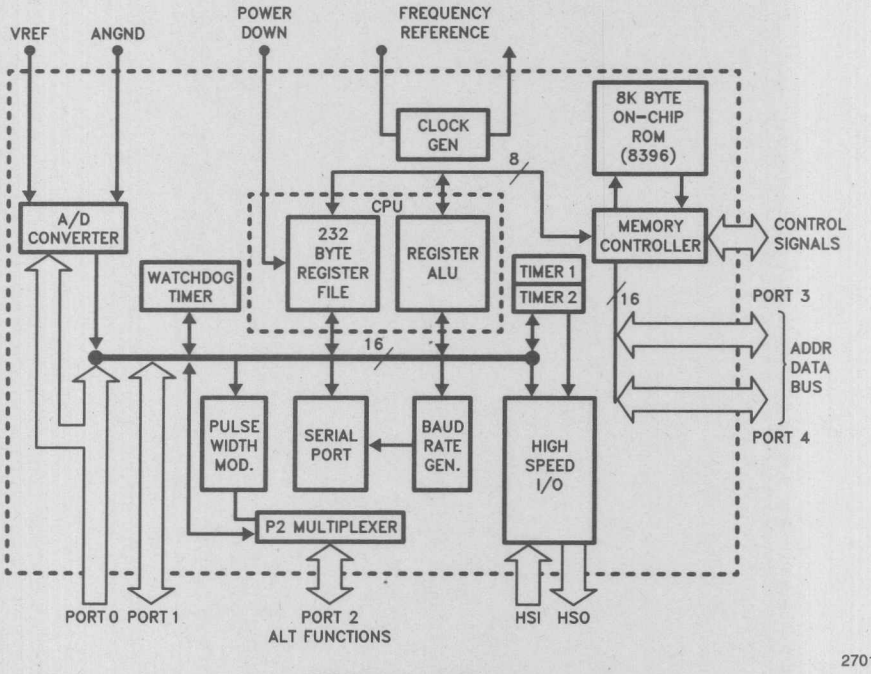
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with $V_{CC} = 5.5V \pm 0.5V$, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



270104-1

MCS-96 Block Diagram

November 1985
Order Number: 270104-001

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . −40°C to +85°C
Storage Temperature ..... −40°C to +150°C
Voltage from Any Pin to
VSS or ANGND ..... −0.3V to +7.0V
Average Output Current from Any Pin .....10 mA
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	−40	+85	°C
V_{CC}	Digital Supply Voltage	4.5	5.5	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.5	5.5	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS T_A = −40°C to +85°C

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage, NMI, XTAL1, $\overline{RESET}$	2.4	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.5	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	V_{PD} Supply Current		1	mA	Normal Operation and Power-Down
I_{REF}	V_{REF} Supply Current		10	mA	
I_{LI}	Input Leakage Current to All Pins of HSI, P0, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to All Pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$		−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_s	Pin Capacitance (Any Pin to V_{SS})		10	pF	f_{TEST} = 1 MHz

NOTES:
1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RSD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

Resolution ..... $\pm 0.001\ V_{REF}$
Accuracy ..... $\pm 0.004\ V_{REF}$
Differential nonlinearity ..... $\pm 0.002\ V_{REF}$ max
Integral nonlinearity ..... $\pm 0.004\ V_{REF}$ max

A.C. CHARACTERISTICS

V_{CC}, V_{PD} = 4.5V to 5.5V, T_A = −40°C to +85°C; f_{osc} = 6.0 MHz to 12.0 MHz
Test Conditions: Load capacitance on output pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS Other system components must meet these specs

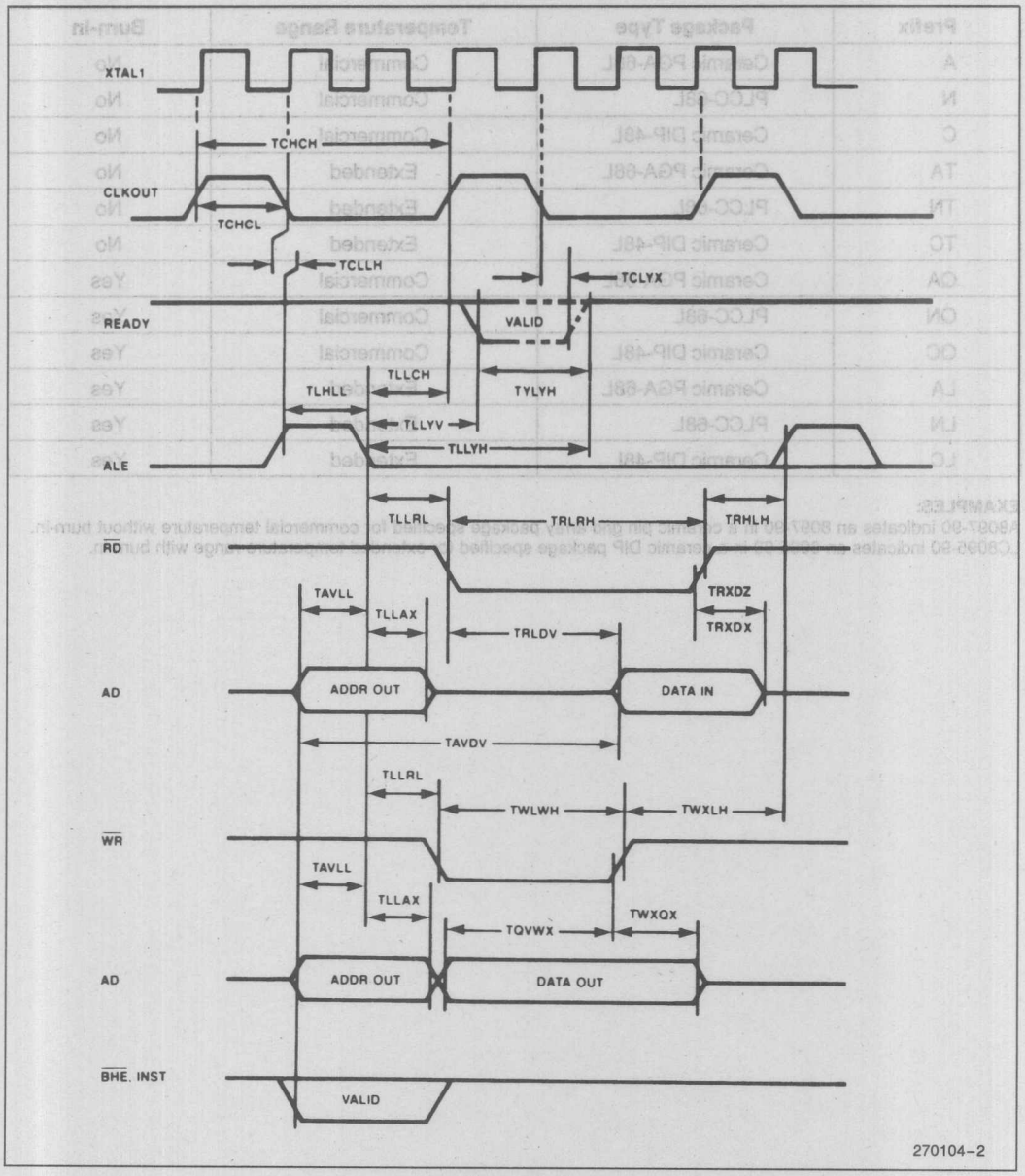
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Falling Edge	0 (Note 1)		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc − 60	ns
TLLYH	End of ALE to READY High	2Tosc + 60	4Tosc − 60 (Note 2)	ns
TYLYH	Non-Ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc − 90	ns
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc − 60	ns
TRXDX	Data Hold after $\overline{RD}$ Inactive (Note 3)	0		ns
TRXDZ	$\overline{RD}$ Inactive to Input Data Float (Note 3)		Tosc − 20	ns

TIMING RESPONSES MCS-96 parts meet these specs

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TCHCH	CLKOUT Period (Note 3)	3Tosc (Note 4)	3Tosc (Note 4)	ns
TCHCL	CLKOUT High Time	Tosc − 20	Tosc + 20	ns
TCLLH	CLKOUT Low to ALE High	−10	30	ns
TLLCH	ALE Low to CLKOUT High	Tosc − 20	Tosc + 40	ns
TLHLL	ALE Pulse Width	Tosc − 25	Tosc + 20	ns
TAVLL	Address Setup to End of ALE	Tosc − 50		ns
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc − 20		ns
TLLAX	Address Hold after End of ALE	Tosc − 20		ns
TWLWH	$\overline{WR}$ Pulse Width	2Tosc − 35		ns
TQVWX	Output Data Setup to End of $\overline{WR}$	2Tosc − 60		ns
TWXQX	Output Data Hold after End of $\overline{WR}$	Tosc − 25		ns
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc − 30		ns
TRLRH	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc − 30		ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc + 60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns.

WAVEFORM

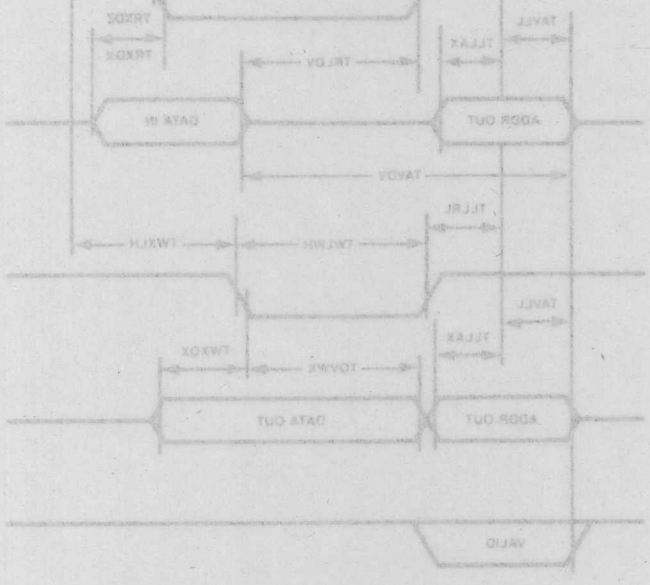


Bus Signal Timings

Table 1. MCS®-96 Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
A	Ceramic PGA-68L	Commercial	No
N	PLCC-68L	Commercial	No
C	Ceramic DIP-48L	Commercial	No
TA	Ceramic PGA-68L	Extended	No
TN	PLCC-68L	Extended	No
TC	Ceramic DIP-48L	Extended	No
QA	Ceramic PGA-68L	Commercial	Yes
QN	PLCC-68L	Commercial	Yes
QC	Ceramic DIP-48L	Commercial	Yes
LA	Ceramic PGA-68L	Extended	Yes
LN	PLCC-68L	Extended	Yes
LC	Ceramic DIP-48L	Extended	Yes

EXAMPLES:
A8097-90 indicates an 8097-90 in a ceramic pin grid array package specified for commercial temperature without burn-in.
LC8095-90 indicates an 8095-90 in a ceramic DIP package specified for extended temperature range with burn-in.



80C196KB
16-BIT HIGH PERFORMANCE CHMOS MICROCONTROLLER

- 232 Byte Register File
- Register-to-Register Architecture
- 28 Interrupt Sources/16 Vectors
- 2.3 μs 16 x 16 Multiply (12 MHz)
- 4.0 μs 32/16 Divide (12 MHz)
- Powerdown and Idle Modes
- Five 8-Bit I/O Ports
- 16-Bit Watchdog Timer
- Dynamically Configurable 8-Bit or 16-Bit Buswidth
- Full Duplex Serial Port
- High Speed I/O Subsystem
- 16-Bit Timer
- 16-Bit Up/Down Counter with Capture
- Pulse-Width-Modulated Output
- Four 16-Bit Software Timers
- 10-Bit A/D Converter with S/H
- $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ Bus Protocol
- 12 MHz Version — 80C196KB12
  10 MHz Version — 80C196KB10

The 80C196KB 16-bit microcontroller is a high performance member of the MCS®-96 microcontroller family. The 80C196KB is pin-for-pin compatible and uses a true superset of the 8096 instructions. Intel's CHMOS process provides a high performance processor along with low power consumption. To further reduce power requirements, the processor can be placed into Idle or Powerdown Mode.

Bit, byte, word and some 32-bit operations are available on the 80C196KB. With a 12 MHz oscillator a 16-bit addition takes 0.66 μs, and the instruction times average 0.5 μs to 1.5 μs in typical applications.

Four high-speed capture inputs are provided to record times when events occur. Six high-speed outputs are available for pulse or waveform generation. The high-speed output can also generate four software timers or start an A/D conversion. Events can be based on the timer or up/down counter.

Also provided on-chip are an A/D converter, serial port, watchdog timer, and a pulse-width-modulated output signal.

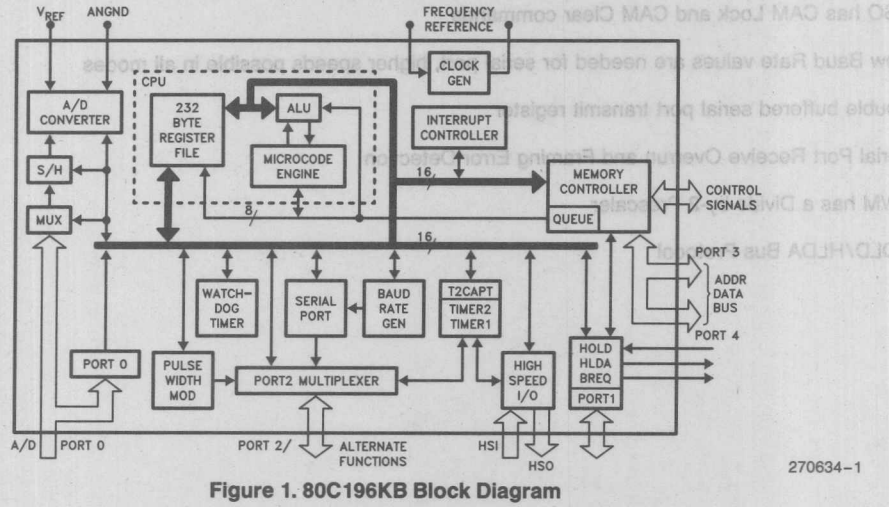


Figure 1. 80C196KB Block Diagram

MCS®-96 is a registered trademark of Intel Corporation.

October 1988
Order Number: 270634-001

ARCHITECTURE

The 80C196KB is a member of the MCS®-96 family, and as such has the same architecture and uses the same instruction set as the 8096. Many new features have been added on the 80C196KB including:

CPU FEATURES

Divide by 2 instead of divide by 3 clock for 1.5X performance

Faster instructions, especially indexed/indirect data operations

2.33 μs 16 $\times$ 16 multiply with 12 MHz clock (was 6.25 μs) on the 8096

Faster interrupt response (almost twice as fast as 8096)

Powerdown and Idle Modes

Clock Failure Detect

6 new instructions including Compare Long and Block Move

8 new interrupt vectors/6 new interrupt sources

PERIPHERAL FEATURES

SFR Window switching allows read-only registers to be written and vice-versa

Timer2 can count up or down by external selection

Timer2 has an independent capture register

HSO line events are stored in a register

HSO has CAM Lock and CAM Clear commands

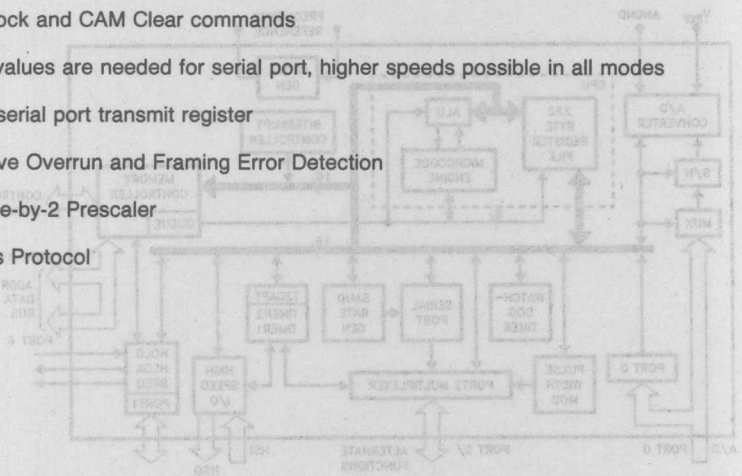
New Baud Rate values are needed for serial port, higher speeds possible in all modes

Double buffered serial port transmit register

Serial Port Receive Overrun and Framing Error Detection

PWM has a Divide-by-2 Prescaler

HOLD/HLDA Bus Protocol



PUSHA — PUSHes the PSW, IMASK, IMASK1, and WSR
(Used instead of PUSHF when new interrupts and registers are used.)

assembly language format: PUSHA
object code format: <11110100>
bytes: 1
states: on-chip stack: 12
off-chip stack: 18

POPA — POPs the PSW, IMASK, IMASK1, and WSR
(Used instead of POPF when new interrupts and registers are used.)

assembly language format: POPA
object code format: <11110101>
bytes: 1
states: on-chip stack: 12
off-chip stack:18

IDLPD — Sets the part into Idle or Powerdown Mode

assembly language format: IDLPD #key (key=1 for Idle, key=2 for Powerdown.)
object code format: <11110110> <key>
bytes: 2
states: legal key: 8
illegal key: 25

DJNZW* — Decrement Jump Not Zero using a Word counter

assembly language format: DJNZW wreg, cadd
object code format: <11100001> <wreg> <disp>
bytes: 3
states: jump not taken: 6
jump taken: 10

CMPL — Compare 2 long direct values

assembly language format:

```
                DST  SRC
CMPL            Lreg, Lreg
```

object code format: <11000101> <src Lreg> <dst Lreg>
bytes: 3
states: 7

BMOV — Block move using 2 auto-incrementing pointers and a counter

assembly language format:

```
                PTRS  CNTREG
BMOV            Lreg, wreg
```

object code format: <11000001> <wreg> <Lreg>
bytes: 3
states: internal/internal: 8 per transfer + 6
external/internal: 11 per transfer + 6
external/external: 14 per transfer + 6

*The DJNZW instruction is not guaranteed to work. See the Functional Deviations section.

SFR OPERATION

All of the registers that were present on the 8096 work the same way as they did, except that the baud rate value is different. The new registers shown in the memory map control new functions. The most important new register is the Window Select Register (WSR) which allows reading of the formerly write-only registers and vice-versa. Using the WSR is described later in this data sheet.

PACKAGING

The 80C196KB is available in a 68-pin PLCC package. Contact your local sales office to determine the exact ordering code for the part desired.

PLCC	Description	PLCC	Description	PLCC	Description
9	ACH7/P0.7	54	AD6/P3.6	31	P1.6/$\overline{\text{HLDA}}$
8	ACH6/P0.6	53	AD7/P3.7	30	P1.5/$\overline{\text{BREQ}}$
7	ACH2/P0.2	52	AD8/P4.0	29	HSO.1
6	ACH0/P0.0	51	AD9/P4.1	28	HSO.0
5	ACH1/P0.1	50	AD10/P4.2	27	HSO.5/HSI.3
4	ACH3/P0.3	49	AD11/P4.3	26	HSO.4/HSI.2
3	NMI	48	AD12/P4.4	25	HSI.1
2	$\overline{\text{EA}}$	47	AD13/P4.5	24	HSI.0
1	V_{CC}	46	AD14/P4.6	23	P1.4
68	V_{SS}	45	AD15/P4.7	22	P1.3
67	XTAL1	44	T2CLK/P2.3	21	P1.2
66	XTAL2	43	READY	20	P1.1
65	CLKOUT	42	T2RST/P2.4/$\overline{\text{AINC}}$	19	P1.0
64	BUSWIDTH	41	$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	18	TXD/P2.0
63	INST	40	$\overline{\text{WR}}$/$\overline{\text{WRL}}$	17	RXD/P2.1
62	ALE/$\overline{\text{ADV}}$	39	PWM/P2.5	16	$\overline{\text{RESET}}$
61	$\overline{\text{RD}}$	38	P2.7/T2CAPTURE/$\overline{\text{PACT}}$	15	EXTINT/P2.2
60	AD0/P3.0	37	V_{PP}	14	CDE(1)
59	AD1/P3.1	36	V_{SS}	13	V_{REF}
58	AD2/P3.2	35	HSO.3/SID3	12	ANGND
57	AD3/P3.3	34	HSO.2/SID2	11	ACH4/P0.4
56	AD4/P3.4	33	P2.6/T2UP-DN	10	ACH5/P0.5
55	AD5/P3.5	32	P1.7/$\overline{\text{HOLD}}$		

Figure 2. Pin Definitions

NOTE:
1. The CDE function is not guaranteed to work. To ensure proper 80C196KB operation, the pin must be grounded.

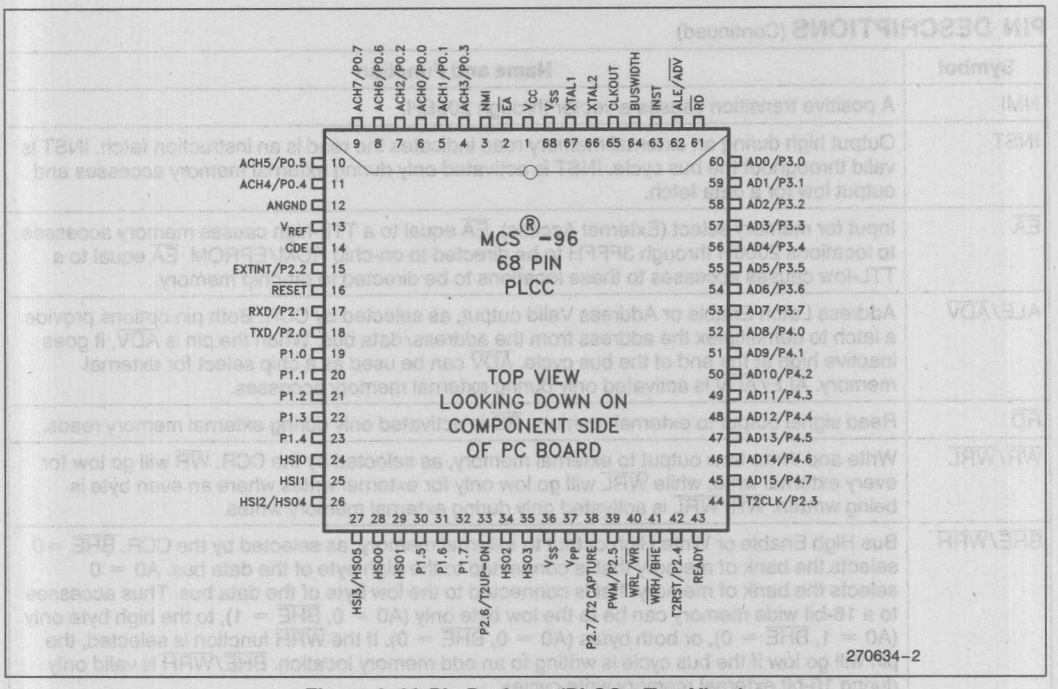


Figure 3. 68-Pin Package (PLCC—Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
CDE(1)	Clock Detect Enable - When pulled high enables the clock failure detection circuit. If the XTAL1 frequency falls below a specified limit the $\overline{\text{RESET}}$ pin will be pulled low.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Timing pin for the return from powerdown circuit. Connect this pin with a 1 μF capacitor to V_{SS} and a 1 MΩ resistor to V_{CC}. If this function is not used V_{PP} may be tied to V_{CC}. This pin was V_{BB} on the 8X9X-90 parts and is the programming voltage on EPROM part.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ½ the oscillator frequency. It has a 50% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 4 state times to reset the chip. The subsequent low-to-high transition re- synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
NMI	A positive transition causes a vector through 203EH.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses and output low for a data fetch.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.
$\overline{WR}$/$\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}$/$\overline{WRL}$ is activated only during external memory writes.
$\overline{BHE}$/$\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE} = 0$ selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{BHE}$/$\overline{WRH}$ is valid only during 16-bit external memory write cycles.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as the SID in Slave Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. Three pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins set the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. All of its pins are shared with other functions in the 80C196KB.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Available only on future ROM and EPROM parts.
$\overline{HOLD}$	Bus Hold input requesting control of the bus. Enabled by setting WSR.7.
$\overline{HLDA}$	Bus Hold acknowledge output indicating release of the bus. Enabled by setting WSR.7.
$\overline{BREQ}$	Bus Request output activated when the bus controller has a pending external memory cycle. Enabled by setting WSR.7.

NOTE:
1. The CDE function is not guaranteed to work. To ensure proper 80C196KB operation, the pin must be grounded.

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	–	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	–	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	–	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	–	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	–	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	
AND/ANDB	2	D ← D AND A	✓	✓	0	0	–	–	
AND/ANDB	3	D ← B AND A	✓	✓	0	0	–	–	
OR/ORB	2	D ← D OR A	✓	✓	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	✓	✓	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP − 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC ← (SP); SP ← SP + 2	–	–	–	–	–	–	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	–	–	–	–	–	–	5
JC	1	Jump if C = 1	–	–	–	–	–	–	5
JNC	1	jump if C = 0	–	–	–	–	–	–	5
JE	1	jump if Z = 1	–	–	–	–	–	–	5
JNE	1	Jump if Z = 0	–	–	–	–	–	–	5
JGE	1	Jump if N = 0	–	–	–	–	–	–	5
JLT	1	Jump if N = 1	–	–	–	–	–	–	5
JGT	1	Jump if N = 0 and Z = 0	–	–	–	–	–	–	5
JLE	1	Jump if N = 1 or Z = 1	–	–	–	–	–	–	5
JH	1	Jump if C = 1 and Z = 0	–	–	–	–	–	–	5
JNH	1	Jump if C = 0 or Z = 1	–	–	–	–	–	–	5
JV	1	Jump if V = 0	–	–	–	–	–	–	5
JNV	1	Jump if V = 1	–	–	–	–	–	–	5
JVT	1	Jump if VT = 1; Clear VT	–	–	–	–	0	–	5
JNVT	1	Jump if VT = 0; Clear VT	–	–	–	–	0	–	5
JST	1	Jump if ST = 1	–	–	–	–	–	–	5
JNST	1	Jump if ST = 0	–	–	–	–	–	–	5
JBS	3	Jump if Specified Bit = 1	–	–	–	–	–	–	5,6
JBC	3	Jump if Specified Bit = 0	–	–	–	–	–	–	5,6
DJNZ/ DJNZW	1	D ← D − 1; If D ≠ 0 then PC ← PC + 8-bit offset	–	–	–	–	–	–	5 10
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	–	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	–	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	–	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	–	–	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↕	↕	0	0	–	–	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	–	–	
CLR/CLRB	1	D ← 0	1	0	0	0	–	–	
SHL/SHLB/SHLL	2	C ← msb - - - - - lsb ← 0	↕	↕	↕	↕	↑	–	7
SHR/SHRB/SHRL	2	0 → msb - - - - - lsb → C	↕	↕	↕	0	–	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb - - - - - lsb → C	↕	↕	↕	0	–	↕	7
SETC	0	C ← 1	–	–	1	–	–	–	
CLRC	0	C ← 0	–	–	0	–	–	–	

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	0	VT ← 0	–	–	–	–	0	–	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interupts (I ← 0)	–	–	–	–	–	–	
EI	0	Enable All Interupts (I ← 1)	–	–	–	–	–	–	
NOP	0	PC ← PC + 1	–	–	–	–	–	–	
SKIP	0	PC ← PC + 2	–	–	–	–	–	–	
NORML	2	Left shift till msb = 1; D ← shift count	↙	↙	0	–	–	–	7
TRAP	0	SP ← SP − 2; (SP) ← PC; PC ← (2010H)	–	–	–	–	–	–	9
PUSHA	1	SP ← SP-2; (SP) ← PSW; PSW ← 0000H; SP ← SP-2; (SP) ← IMASK1/WSR; IMASK1 ← 00H	0	0	0	0	0	0	
POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	↙	↙	↙	↙	↙	↙	
IDLPD	1	IDLE MODE IF KEY=1; POWERDOWN MODE IF KEY =2; CHIP RESET OTHERWISE	–	–	–	–	–	–	
CMPL	2	D-A	↙	↙	↙	↙	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

NOTES:

1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is done. Operands is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D,D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D,D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.
10. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

Instruction Execution State Times (Minimum) (1)

MNEMONIC	DIRECT	IMMED	INDIRECT		INDEXED	
			NORMAL*	A-INC*	SHORT*	LONG*
ADD (3-op)	5	6	7/10	8/11	7/10	8/11
SUB (3-op)	5	6	7/10	8/11	7/10	8/11
ADD (2-op)	4	5	6/8	7/9	6/8	7/9
SUB (2-op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3-op)	5	5	7/10	8/11	7/10	8/11
SUBB (3-op)	5	5	7/10	8/11	7/10	8/11
ADDB (2-op)	4	4	6/8	7/9	6/8	7/9
SUBB (2-op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
MUL (3-op)	16	17	18/21	19/22	19/22	20/23
MULU (3-op)	14	15	16/19	17/19	17/20	18/21
MUL (2-op)	16	17	18/21	19/22	19/22	20/23
MULU (2-op)	14	15	16/19	17/19	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3-op)	12	12	14/17	15/18	15/18	16/19
MULUB (3-op)	10	10	12/15	13/15	12/16	14/17
MULB (2-op)	12	12	14/17	15/18	15/18	16/19
MULUB (2-op)	10	10	12/15	13/15	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3-op)	5	6	7/10	8/11	7/10	8/11
AND (2-op)	4	5	6/8	7/9	6/8	7/9
OR (2-op)	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3-op)	5	5	7/10	8/11	7/10	8/11
ANDB (2-op)	4	4	6/8	7/9	6/8	7/9
ORB (2-op)	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD/LDB	4	5	5/8	6/8	6/9	7/10
ST/STB	4	5	5/8	6/9	6/9	7/10
LDBSE	4	4	5/8	6/8	6/9	7/10
LDBZE	4	4	5/8	6/8	6/9	7/10
BMOV	6+8 per word			6 + 11/14 per word		
PUSH (int stack)	6	7	9/12	10/13	10/13	11/14
POP (int stack)	8	–	10/12	11/13	11/13	12/14
PUSH (ext stack)	8	9	11/14	12/15	12/15	13/16
POP (ext stack)	11	–	13/15	14/16	14/16	15/17

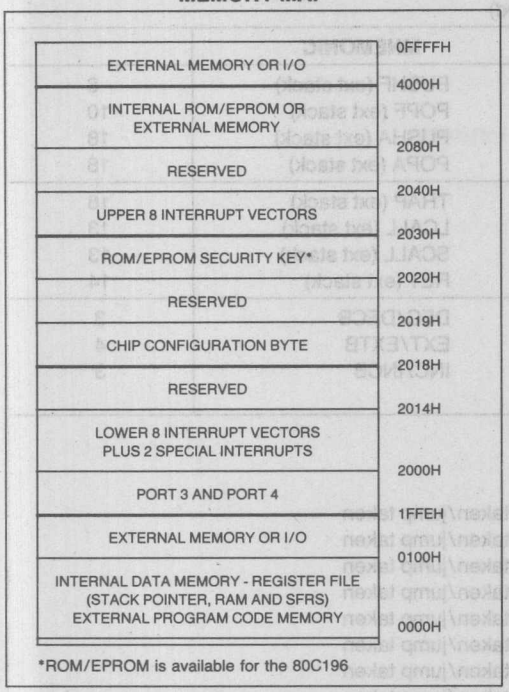
*Times for (Internal/External) Operands

NOTE:
1. Execution times for instructions accessing external data memory may be one to two states higher depending on the instruction stream being executed. In sixteen bit mode, the minimum execution state times apply for instructions accessing internal register space. Execution times do not reflect eight bit mode or insertion of wait states.

MNEMONIC		MNEMONIC	
PUSHF (int stack)	6	PUSHF (ext stack)	8
POPF (int stack)	7	POPF (ext stack)	10
PUSHA (int stack)	12	PUSHA (ext stack)	18
POPA (int stack)	12	POPA (ext stack)	18
TRAP (int stack)	16	TRAP (ext stack)	18
LCALL (int stack)	11	LCALL (ext stack)	13
SCALL (int stack)	11	SCALL (ext stack)	13
RET (int stack)	11	RET (ext stack)	14
CMPL	7	DEC/DECB	3
CLR/CLRB	3	EXT/EXTB	4
NOT/NOTB	3	INC/INCB	3
NEG/NEGB	3		
LJMP	7		
SJMP	7		
BR [indirect]	7		
JNST, JST	4/8 jump not taken/jump taken		
JNH, JH	4/8 jump not taken/jump taken		
JGT, JLE	4/8 jump not taken/jump taken		
JNC, JC	4/8 jump not taken/jump taken		
JNVT, JVT	4/8 jump not taken/jump taken		
JNV, JV	4/8 jump not taken/jump taken		
JGE, JLT	4/8 jump not taken/jump taken		
JNE, JE	4/8 jump not taken/jump taken		
JBC, JBS	5/9 jump not taken/jump taken		
DJNZ	5/9 jump not taken/jump taken		
DJNZW (Note 1)	5/9 jump not taken/jump taken		
NORML	8 + 1 per shift (9 for 0 shift)		
SHRL	7+ 1 per shift (8 for 0 shift)		
SHLL	7 + 1 per shift (8 for 0 shift)		
SHRAL	7 + 1 per shift (8 for 0 shift)		
SHR/SHRB	6 + 1 per shift (7 for 0 shift)		
SHL/SHLB	6 + 1 per shift (7 for 0 shift)		
SHRA/SHRAB	6 + 1 per shift (7 for 0 shift)		
CLRC	2		
SETC	2		
DI	2		
EI	2		
CLRVT	2		
NOP	2		
RST	15 (includes fetch of configuration byte)		
SKIP	3		
IDLPD	8/25 (proper key/improper key)		

NOTE:
1. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

MEMORY MAP



*ROM/EPROM is available for the 80C196

80C196KB INTERRUPTS

Number	Source	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT Pin	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

	WHEN READ	WHEN WRITTEN
19H	STACK POINTER	STACK POINTER
18H		
17H	*IOS2	PWM_CONTROL
16H	IOS1	IOC1
15H	IOS0	IOC0
14H	*WSR	*WSR
13H	*INT_MASK 1	*INT_MASK 1
12H	*INT_PEND 1	*INT_PEND 1
11H	*SP_STAT	*SP_CON
10H	PORT2	PORT2
0FH	PORT1	PORT1
0EH	PORT0	BAUD RATE
0DH	TIMER2 (HI)	TIMER2 (HI)
0CH	TIMER2 (LO)	TIMER2 (LO)
0BH	TIMER1 (HI)	*IOC2
0AH	TIMER1 (LO)	WATCHDOG
09H	INT_PENDING	INT_PENDING
08H	INT_MASK	INT_MASK
07H	SBUF(RX)	SBUF(TX)
06H	HSI_STATUS	HSO_COMMAND
05H	HSI_TIME (HI)	HSO_TIME (HI)
04H	HSI_TIME (LO)	HSO_TIME (LO)
03H	AD_RESULT (HI)	HSI_MODE
02H	AD_RESULT (LO)	AD_COMMAND
01H	ZERO REG (HI)	ZERO REG (HI)
00H	ZERO REG (LO)	ZERO REG (LO)

WSR = 0

	WSR = 15
0FH	RESERVED (1)
0EH	RESERVED (1)
0DH	*T2 CAPTURE (HI)
0CH	*T2 CAPTURE (LO)

OTHER SFRS IN WSR 15 BECOME READABLE IF THEY WERE WRITABLE IN WSR = 0 AND WRITABLE IF THEY WERE READABLE IN WSR = 0

*NEW OR CHANGED REGISTER FUNCTION

NOTE:
1. Reserved registers should not be written.

USING THE ALTERNATE REGISTER WINDOW (WSR = 15)

I/O register expansion on the new CHMOS members of the MCS-96 family has been provided by making two register windows available. Switching between these windows is done using the Window Select Register (WSR). The PUSHA and POPA instructions can be used to push and pop the WSR and second interrupt mask when entering or leaving interrupts, so it is easy to change between windows.

On the 80C196KB only Window 0 and Window 15 are active. Window 0 is a true superset of the standard 8096 SFR space, while Window 15 allows the read-only registers to be written and write-only registers to be read. The only major exception to this is the Timer2 register which is the Timer2 capture register in Window 15. The writeable register for Timer2 is in Window 0. There are also some minor changes and cautions. The descriptions of the registers which have different functions in Window 15 than in Window 0 are listed below:

AD_COMMAND (02H) — Read the last written command

AD_RESULT (02H, 03H) — Write a value into the result register

HSI_MODE (03H) — Read the value in HSI_MODE

HSI_TIME (04H,05H) — Write to FIFO Holding register

HSO_TIME (04H,05H) — Read the last value placed in the holding register

HSI_STATUS (06H) — Write to status bits but not to HSI pin bits. (Pin bits are 1,3,5,7).

HSO_COMMAND (06H) — Read the last value placed in the holding register

SBUF(RX) (07H) — Write a value into the receive buffer

SBUF(TX) (07H) — Read the last value written to the transmit buffer

WATCHDOG(0AH) — Read the value in the upper byte of the WDT

TIMER1 (0AH,0BH) — Write a value to Timer1

TIMER2 (0CH,0DH) — Read/Write the Timer2 capture register.
Note that Timer2 read/write is done with WSR = 0.

IOC2 (0BH) — Last written value is readable, except bit 7 (note 1)

BAUD_RATE (0EH) — No function, cannot be read

PORT0 (0EH) — No function, no output drivers on the pins. Register reserved.

PORT1 — IOPORT1 cannot be read or written in Window 15. Register reserved.

SP_STAT (11H) — Set the status bits, TI and RI can be set, but it will not cause an interrupt

SP_CON (11H) — Read the current control byte

IOS0 (15H) — Writing to this register controls the HSO pins. Bits 6 and 7 are inactive for writes.

IOC0 (15H) — Last written value is readable, except bit 1 (note 1)

IOS1 (16H) — Writing to this register will set the status bits, but not cause interrupts. Bits 6 and 7 are not functional

IOC1 (16H) — Last written value is readable

IOS2 (17H) — Writing to this register will set the status bits, but not cause interrupts.

PWM_CONTROL (17H) — Read the duty cycle value written to PWM_CONTROL

NOTE:

1. IOC2.7 (CAM CLEAR) and IOC0.1 (T2RST) are not latched and will read as a 1 (precharged bus) .

Being able to write to the read-only registers and vice-versa provides a lot of flexibility. One of the most useful advantages is the ability to set the timers and HSO lines for initial conditions other than zero.

Reserved registers may be used for testing as future features. Do not write to these registers. Read from reserved registers will return indeterminate values.

SFR BIT SUMMARY

A summary of the SFRs which control I/O functions has been included in this section. The summary is separated into a list of those SFRs which have changed on the 80C196KB and a list of those which have remained almost the same.

The following 80C196KB SFRs are different than those on the 8096BH:

(The Read and Write comments indicate the register's function in Window 0 unless otherwise specified.)

SBUF(TX): Now double buffered
07h
write

BAUD RATE: Uses new Baud Rate Values
0Eh
write

SP_STAT:

7	6	5	4	3	2	1	0
RB8/ RPE	RI	TI	FE	TXE	OE	X	X

11h
read

RPE : Receive Parity Error
RI : Receive Indicator
TI : Transmit Indicator
FE : Framing Error
TXE : Transmitter Empty
OE : Receive Overrun Error

IPEND1:
IMASK1:

7	6	5	4	3	2	1	0
NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI

12h,13h
read/write

NMI : Non-Maskable Interrupt (set to 0 for future compatibility)
FIFO FULL : HSIO FIFO full
EXTINT : External Interrupt Pin
T2OVF : Timer2 Overflow
T2CAP : Timer2 Capture
HSI4 : HSI has 4 or more entries in FIFO
RI : Receive Interrupt
TI : Transmit Interrupt

WSR:

7	6	5	4	3	2	1	0
HLDEN	0	0	0	W	W	W	W

14h
read/write

WWWW = 0 : SFRs function like a superset of 8096 SFRs
WWWW = 14 : PPW register
WWWW = 15 : Exchange read/write registers
WWWW = OTHER : Undefined, do not use
000 : These bits must always be written as zeros to provide compatibility with future products.
HLDEN = 1 : Enables the HOLD/HLDA bus protocol

IOS2:

7	6	5	4	3	2	1	0
START A2D	T2 RESET	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

17h
read

Indicates which HSO event occured
START A2D : HSO__CMD 15, start A to D
T2RESET : HSO__CMD 14, Timer 2 reset
HSO.0-5 : Output pins HSO.0 through HSO.5

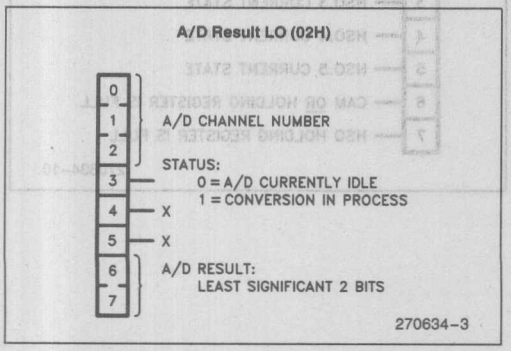
IOC2:

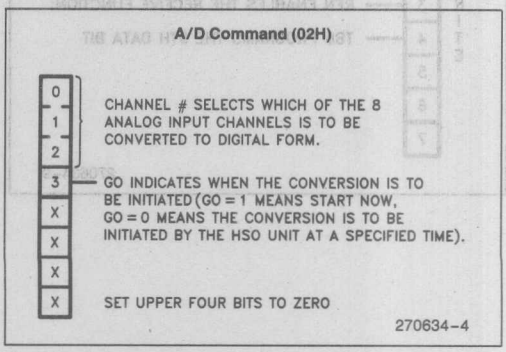
7	6	5	4	3	2	1	0
CLEAR CAM	ENA LOCK	T2ALT INT	A2D CPD	X	SLOW PWM	T2UD ENA	FAST T2EN

0Bh
write

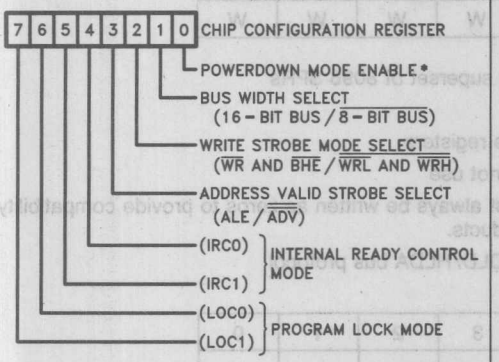
CLEAR__CAM : Clear Entire CAM
ENA__LOCK : Enable lockable CAM entry feature
T2ALT INT : Enable T2 Alternate Interrupt at 8000H
A2D__CPD : Clock Prescale Disable for low XTAL frequency (A to D conversion in fewer state times)
X : Set to 0
SLOW__PWM : Turn on divide by 2 Prescaler on PWM
T2UD ENA : Enable Timer 2 as up/down counter
FAST__T2EN : Enable Fast increment of T2; once per state time.

The following registers are the same on the 80C196KB as they were on the 8096BH:





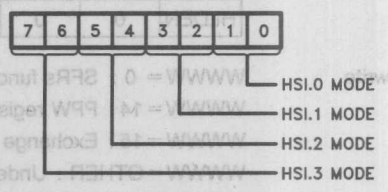
Chip Configuration (2018H)



270634-5

*Minor Change

HSI_Mode (03H)

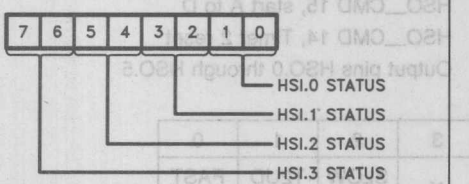


WHERE EACH 2 – BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

270634-6

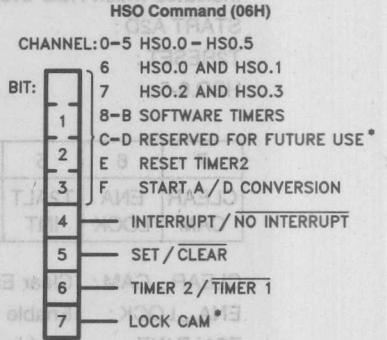
HSI_Status (06H)



WHERE FOR EACH 2 – BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270634-7

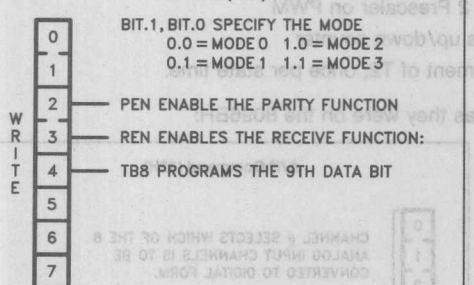
HSO Command (06H)



270634-8

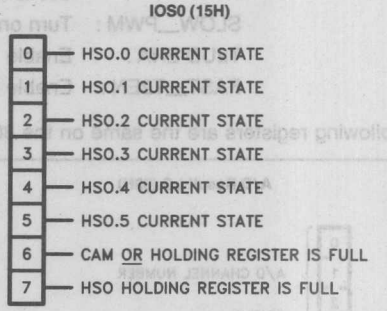
*Minor Change

SPCON (11H)

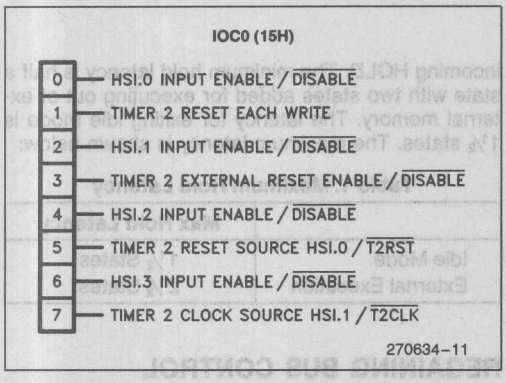


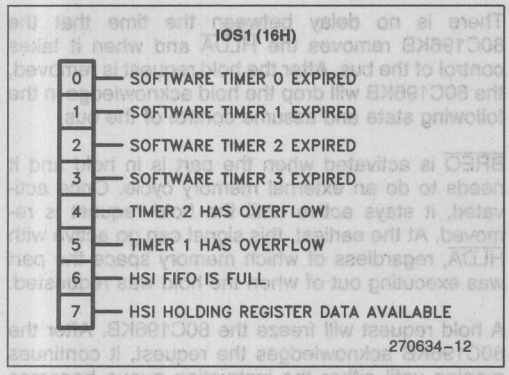
270634-9

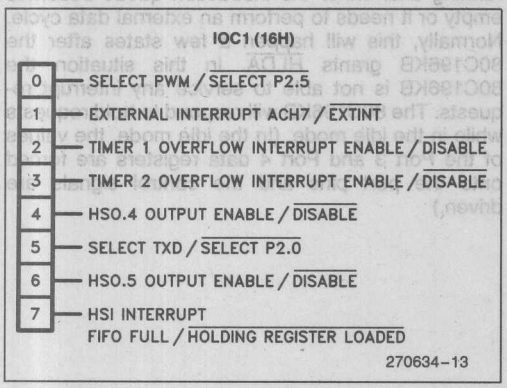
IOS0 (15H)



270634-10







Port 2 Multiple Functions

Pin	Func.	Alternative Function	Control Reg.
2.0	Output	TXD (Serial Port Transmit)	IOC1.5
2.1	Input	RXD (Serial Port Receive)	SPCON.3
2.3	Input	T2CLK (Timer2 Clock & Baud)	IOC0.7
2.4	Input	T2RST (Timer2 Reset)	IOC0.5
2.5	Output	PWM Output	IOC1.0
2.6	QBD*	Timer2 up/ down select	IOC2.1
2.7	QBD*	Timer2 Capture	N/A

*QBD = Quasi-bidirectional

Baud Rate Calculations

Asynchronous Modes 1, 2 and 3:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 16} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate} \times 8}$$

Synchronous Mode 0:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 2} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate}}$$

Baud Rates and Baud Register Values

Baud Rate	XTAL Frequency					
	8.0 MHz		10.0 MHz		12.0 MHz	
300	1666	−0.02	2082	0.02	2499	0.00
1200	416	−0.08	520	−0.03	624	0.00
2400	207	0.16	259	0.16	312	−0.16
4800	103	−0.16	129	0.16	155	0.16
9600	51	−0.16	64	0.16	77	0.16
19.2K	25	0.16	32	1.40	38	0.16

Baud Register Value/% Error

A maximum baud rate of 750 Kbaud is available in the asynchronous modes with 12 MHz on XTAL1. The synchronous mode has a maximum rate of 3.0 Mbaud with a 12 MHz clock. Location 0EH is the Baud Register. It is loaded sequentially in two bytes, with the low byte being loaded first. This register may not be loaded with zero in serial port Mode 0.

NOTE:
The maximum T2CLK rate is 3 MHz when used to set the baud rate.

HOLD/HLDA PROTOCOL

The 80C196KB supports a bus exchange protocol, allowing other devices that are capable of acting as bus masters to gain control of the bus. The hand shake consists of three signals, $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$. $\overline{\text{HOLD}}$, which is an input, is activated by a bus master which requires control of the bus. The 80C196KB responds by releasing the bus and activating the $\overline{\text{HLDA}}$ output line. When the new bus master is finished with the bus, it returns control of the bus to the 80C196KB by dropping its hold request. In response, the 80C196KB removes its hold acknowledge and assumes control of the bus. The third signal, $\overline{\text{BREQ}}$, is activated by the 80C196KB when it is in hold and has a pending external bus cycle. The 80C196KB deactivates this signal at the same time it removes the acknowledge signal.

Figure 4 shows the bus timing for $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$. The $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$ signals are multiplexed with P1.7, P1.6 and P1.5 respectively. To enable the bus hold feature on these pins instead of the Port 1 feature, the HLDEN(WSR.7) must be set to one. This bit is cleared during reset and must be set by software to configure the pins as well as to enable the bus hold feature. Once this bit is set, the three most significant pins of Port1 cannot be restored to their I/O function without resetting the part. The bus hold feature, however, can be disabled by clearing the HLDEN.

When the 80C196KB acknowledges the hold request, the output buffers for Port3, Port4 and all the bus control signals, (ALE, $\overline{\text{BHE}}$, INST, $\overline{\text{RD}}$ and $\overline{\text{WR}}$), are turned off. Although the strong pullup and pulldown on ALE are disabled during hold acknowledge, a weak pulldown is turned on to provide an option to wire OR the ALE with other bus masters. The request to hold acknowledge latency is dependent on the state of the bus controller.

MAXIMUM HOLD LATENCY

The maximum hold latency is the maximum time before the 80C196KB will respond with HLDA to an incoming HOLD. The minimum hold latency is half a state with two states added for executing out of external memory. The latency for exiting idle mode is 1½ states. The maximum latency is shown below:

Table 1. Maximum Hold Latency

	Max Hold Latency
Idle Mode	1½ States
External Execution	2½ States

REGAINING BUS CONTROL

There is no delay between the time that the 80C196KB removes the $\overline{\text{HLDA}}$ and when it takes control of the bus. After the hold request is removed, the 80C196KB will drop the hold acknowledge in the following state and assume control of the bus.

$\overline{\text{BREQ}}$ is activated when the part is in hold and it needs to do an external memory cycle. Once activated, it stays active until the hold request is removed. At the earliest, this signal can go active with $\overline{\text{HLDA}}$, regardless of which memory space the part was executing out of when the hold was requested.

A hold request will freeze the 80C196KB. After the 80C196KB acknowledges the request, it continues running until either the instruction queue becomes empty or it needs to perform an external data cycle. Normally, this will happen a few states after the 80C196KB grants $\overline{\text{HLDA}}$. In this situation the 80C196KB is not able to service any interrupt requests. The 80C196KB will respond to hold requests while in the idle mode. (In the idle mode, the values of the Port 3 and Port 4 data registers are forced onto the port pins and the control signals are driven.)

DISABLING HOLD REQUESTS

HLDEN(WSR.7) can be cleared to block hold requests in cases where consecutive memory cycles are required. Clearing this bit, however, does not cause the 80C196KB to take control of the bus immediately. The part waits until the current hold request is over, then it disables the bus hold feature, causing a new hold request to be ignored until HLDEN is set again. Since there is a delay from the time the code for clearing the bit is fetched to the time the bit is actually cleared, the code that clears HLDEN must be a few instructions ahead of the block that needs to be protected against hold requests. The safest way is to add a JBC instruction to check the status of the HLDA pin after the code that clears the HLDEN bit. A code example is shown in Figure 5. This prevents the part from executing a new instruction until both current hold requests are serviced and the hold feature is disabled.

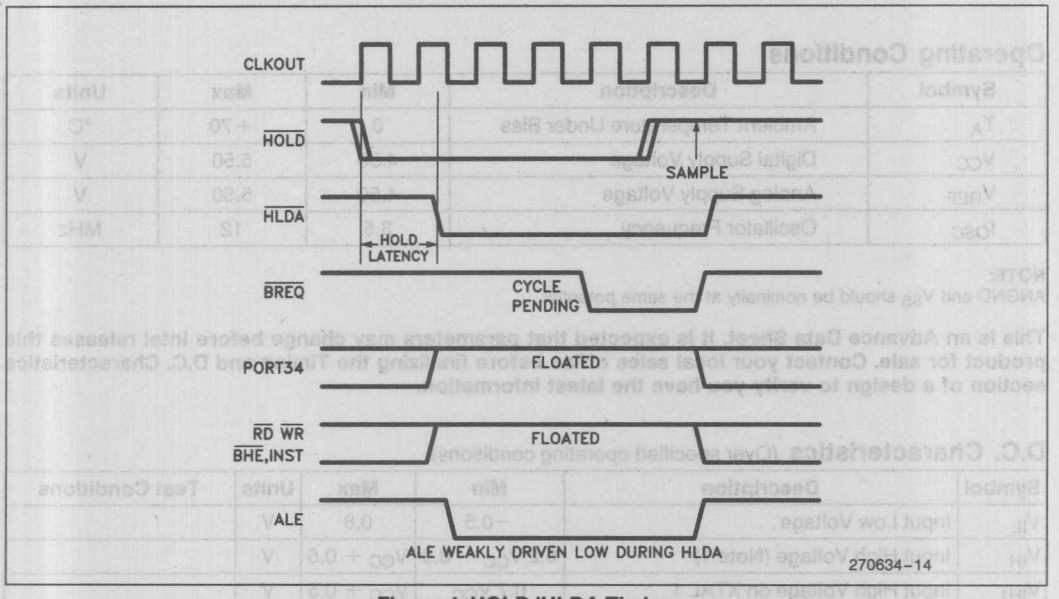


Figure 4. HOLD/HLDA Timing

```
        DI                  ; disable interrupts
        ANDB WSR, #0EFH     ; disable hold request
WAIT:   JBC PORT1, 6, WAIT  ; Check the HLDA pin
          .                 ; If set, execute
          .                 ; protected instructions
          .
        ORB WSR,#80h        ; enable HOLD requests
        EI                  ; enable interrupts
```

NOTE:
Interrupts should be disabled to prevent code interruption

Figure 5. Sample Code For Disabling HOLD/HLDA

ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings*

Ambient Temperature
Under Bias ..... 0°C to +70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin to V_{SS} ..... −0.5V to +7.0V
Power Dissipation 1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

Operating Conditions

Symbol	Description	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	°C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	3.5	12	MHz

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

This is an Advance Data Sheet. It is expected that parameters may change before Intel releases this product for sale. Contact your local sales office before finalizing the Timing and D.C. Characteristics section of a design to verify you have the latest information.

D.C. Characteristics (Over specified operating conditions)

Symbol	Description	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage (Note 1)	$0.2\,V_{CC} + 0.9$	$V_{CC} + 0.5$	V	
V_{IH1}	Input High Voltage on XTAL 1	$0.7\,V_{CC}$	$V_{CC} + 0.5$	V	
V_{IH2}	Input High Voltage on RESET	2.2	$V_{CC} + 0.5$	V	
V_{OL}	Output Low Voltage		0.3 0.45 1.5	V V V	$I_{OL} = 200\ \mu A$ $I_{OL} = 3.2$ mA $I_{OL} = 7$ mA
V_{OH}	Output High Voltage (Standard Outputs)	$V_{CC} - 0.3$ $V_{CC} - 0.7$ $V_{CC} - 1.5$		V V V	$I_{OH} = -200\ \mu A$ $I_{OH} = -3.2$ mA $I_{OH} = -7$ mA
V_{OH1}	Output High Voltage (Quasi-bidirectional Outputs)	$V_{CC} - 0.3$ $V_{CC} - 0.7$ $V_{CC} - 1.5$		V V V	$I_{OH} = -10\ \mu A$ $I_{OH} = -30\ \mu A$ $I_{OH} = -60\ \mu A$
I_{LI}	Input Leakage Current (Std. Inputs)		±10	μA	$0 < V_{IN} < V_{CC} - 0.3V$
I_{LI1}	Input Leakage Current (Port 0)		±3	μA	$0 < V_{IN} < V_{REF}$
I_{TL}	1 to 0 Transition Current (QBD Pins)		−650	μA	$V_{IN} = 2.0V$
I_{IL}	Logical 0 Input Current (QBD Pins)		−50	μA	$V_{IN} = 0.45V$
I_{IL1}	Logical 0 Input Current in Reset (Note 2) (ALE, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, INST, P2.0)		−850	μA	$V_{IN} = 0.45$ V

NOTE:
1. All pins except RESET and XTAL1.
2. Holding these pins below V_{IH} in Reset may cause the part to enter test modes.

D.C. Characteristics (Over specified operating conditions) (Continued)

Symbol	Description	Min	Typ(7)	Max	Units	Test Conditions
I_{CC}	Active Mode Current in Reset		40	55	mA	XTAL1 = 12 MHz
I_{REF}	A/D Converter Reference Current		2	5	mA	$V_{CC} = V_{PP} = V_{REF} = 5.5V$
I_{IDLE}	Idle Mode Current		10	22	mA	
I_{CC1}	Active Mode Current (Typical)		15	22	mA	XTAL1 = 3.5 MHz
I_{PD}(8)	Powerdown Mode Current		5		μA	$V_{CC} = V_{PP} = V_{REF} = 5.5V$
R_{RST}	Reset Pullup Resistor	6K		50K	Ω	
C_S	Pin Capacitance (Any Pin to V_{SS})			10	pF	f_{TEST} = 1.0 MHz

NOTES:
(Notes apply to all specifications)
1. QBD (Quasi-bidirectional) pins include Port 1, P2.6 and P2.7.
2. Standard Outputs include AD0–15, $\overline{RD}$, $\overline{WR}$, ALE, $\overline{BHE}$, INST, HSO pins, PWM/P2.5, CLKOUT, RESET, Ports 3 and 4, TXD/P2.0, and RXD (in serial mode 0). The V_{OH} specification is not valid for RESET. Ports 3 and 4 are open-drain outputs.
3. Standard Inputs include HSI pins, CDE, EA, READY, BUSWIDTH, NMI, RXD/P2.1, EXTINT/P2.2, T2CLK/P2.3, and T2RST/P2.4.
4. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V or V_{OH} is held below V_{CC} − 0.7V:
   I_{OL} on Output pins: 10 mA
   I_{OH} on quasi-bidirectional pins: self limiting
   I_{OH} on Standard Output pins: 10 mA
5. Maximum current per bus pin (data and control) during normal operation is ±3.2 mA.
6. During normal (non-transient) conditions the following total current limits apply:

Port 1, P2.6	I_{OL}: 29 mA	I_{OH} is self limiting
HSO, P2.0, RXD, $\overline{RESET}$	I_{OL}: 29 mA	I_{OH}: 26 mA
P2.5, P2.7, $\overline{WR}$, $\overline{BHE}$	I_{OL}: 13 mA	I_{OH}: 11 mA
AD0–AD15	I_{OL}: 52 mA	I_{OH}: 52 mA
$\overline{RD}$, ALE, INST–CLKOUT	I_{OL}: 13 mA	I_{OH}: 13 mA

7. Typicals are based on a limited number of samples and are not guaranteed. The values listed are at room temperature and $V_{REF} = V_{CC} = 5V$.
8. I_{PD} is not guaranteed on the standard 80C196KB part and may exceed 100 μA on some parts. Customers whose applications use the powerdown mode and require a guaranteed maximum value of I_{PD} should contact an Intel Field Sales Representative.

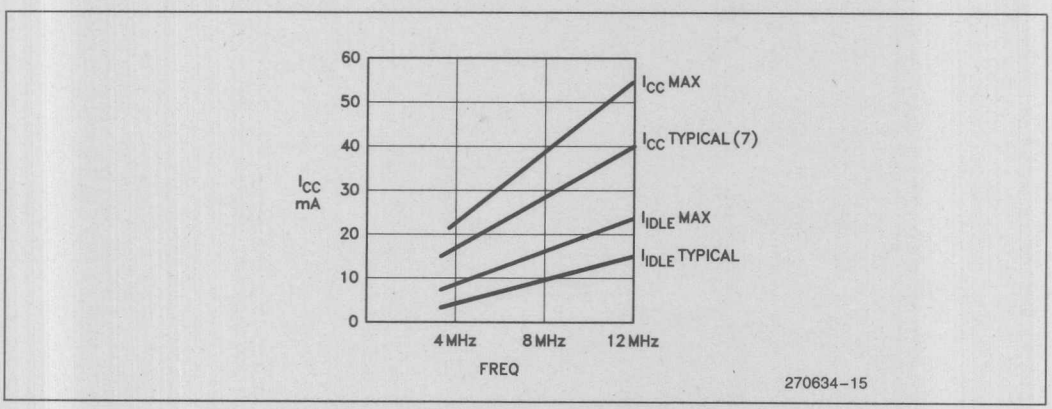


Figure 15. I_{CC} and I_{IDLE} vs Frequency

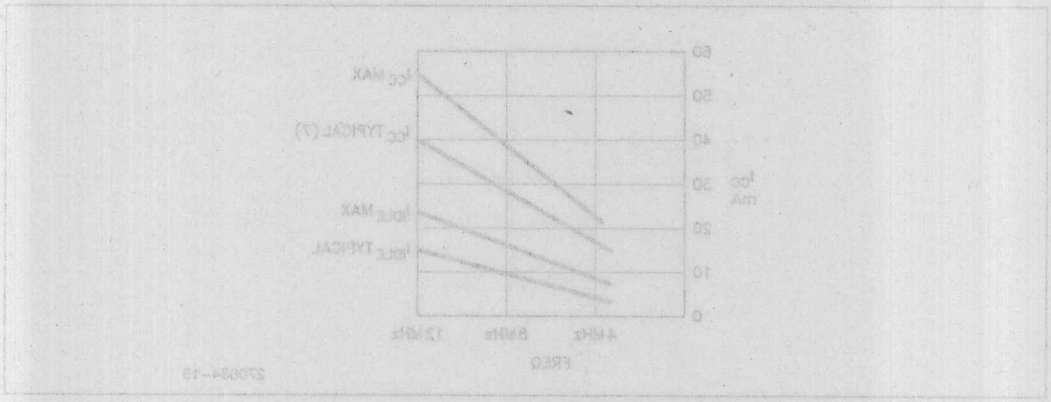
A.C. Characteristics (Over specified operating conditions)

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The system must meet these specifications to work with the 80C196KB:

Symbol	Description	Min	Max	Units	Notes
T_{AVYV}	Address Valid to READY Setup		$2T_{OSC} - 85$	ns	
T_{LLYV}	ALE Low to READY Setup				
	80C196KB10		$T_{OSC} - 80$	ns	
	80C196KB12		$T_{OSC} - 72$	ns	
T_{YLYH}	Non READY Time	No upper limit		ns	
T_{CLYX}	READY Hold after CLKOUT Low	0	$T_{OSC} - 30$	ns	(Note 1)
T_{LLYX}	READY Hold after ALE Low	$T_{OSC} - 15$	$2T_{OSC} - 40$	ns	(Note 1)
T_{AVGV}	Address Valid to Buswidth Setup		$2T_{OSC} - 85$	ns	
T_{LLGV}	ALE Low to Buswidth Setup		$T_{OSC} - 70$	ns	
T_{CLGX}	Buswidth Hold after CLKOUT Low	0		ns	
T_{AVDV}	Address Valid to Input Data Valid				
	80C196KB10		$3T_{OSC} - 70$	ns	
	80C196KB12		$3T_{OSC} - 67$	ns	
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid				
	80C196KB10		$T_{OSC} - 30$	ns	
	80C196KB12		$T_{OSC} - 23$	ns	
T_{CLDV}	CLKOUT Low to Input Data Valid		$T_{OSC} - 50$	ns	
T_{RHDZ}	End of $\overline{RD}$ to Input Data Float		$T_{OSC} - 20$	ns	
T_{RXDX}	Data Hold after $\overline{RD}$ Inactive	0		ns	

NOTES:
1. If max is exceeded, additional wait states will occur.



A.C. Characteristics (Over specified operating conditions) (Continued)

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The 80C196KB will meet these specifications:

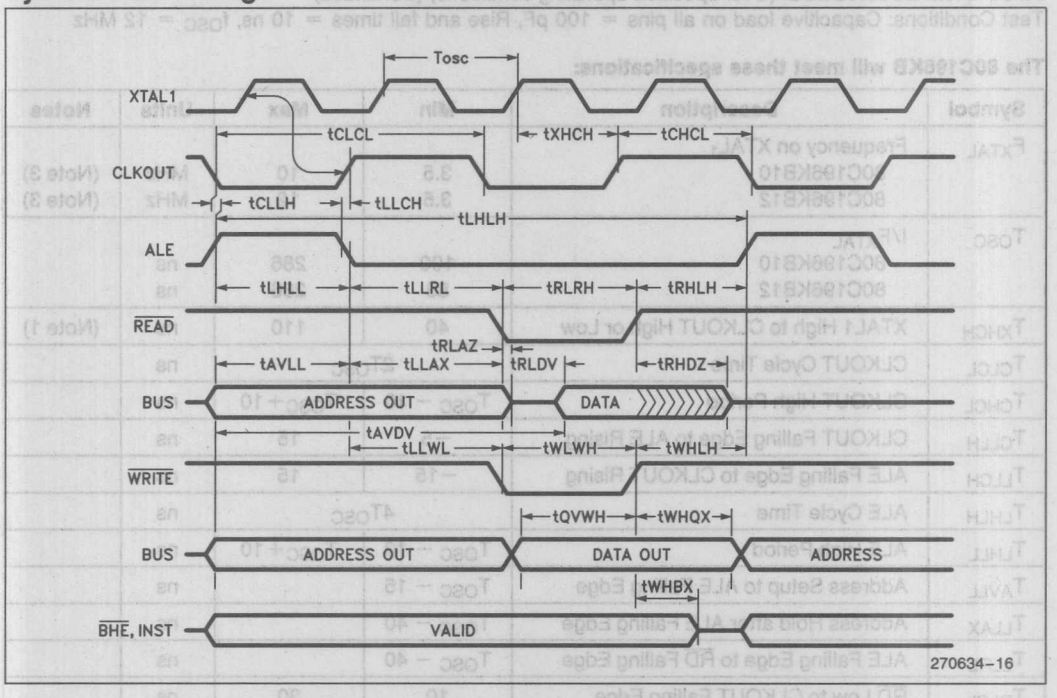
Symbol	Description	Min	Max	Units	Notes
F_{XTAL}	Frequency on $XTAL_1$				
	80C196KB10	3.5	10	MHz	(Note 3)
	80C196KB12	3.5	12	MHz	(Note 3)
T_{OSC}	$1/F_{XTAL}$				
	80C196KB10	100	286	ns	
	80C196KB12	83	286	ns	
T_{XHCH}	XTAL1 High to CLKOUT High or Low	40	110	ns	(Note 1)
T_{CLCL}	CLKOUT Cycle Time	$2T_{OSC}$		ns	
T_{CHCL}	CLKOUT High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{CLLH}	CLKOUT Falling Edge to ALE Rising	−5	15	ns	
T_{LLCH}	ALE Falling Edge to CLKOUT Rising	−15	15	ns	
T_{LHLH}	ALE Cycle Time	$4T_{OSC}$		ns	
T_{LHLL}	ALE High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{AVLL}	Address Setup to ALE Falling Edge	$T_{OSC} - 15$		ns	
T_{LLAX}	Address Hold after ALE Falling Edge	$T_{OSC} - 40$		ns	
T_{LLRL}	ALE Falling Edge to $\overline{RD}$ Falling Edge	$T_{OSC} - 40$		ns	
T_{RLCL}	$\overline{RD}$ Low to CLKOUT Falling Edge	10	30	ns	
T_{RLRH}	$\overline{RD}$ Low Period	$T_{OSC} - 5$		ns	
T_{RHLH}	$\overline{RD}$ Rising Edge to ALE Rising Edge	T_{OSC}	$T_{OSC} + 25$	ns	(Note 2)
T_{RLAZ}	$\overline{RD}$ Low to Address Float		10	ns	
T_{LLWL}	ALE Falling Edge to $\overline{WR}$ Falling Edge	$T_{OSC} - 10$		ns	
T_{CLWL}	CLKOUT Low to $\overline{WR}$ Falling Edge	0	25	ns	
T_{QVWH}	Data Stable to $\overline{WR}$ Rising Edge				
	80C196KB10	$T_{OSC} - 30$		ns	
	80C196KB12	$T_{OSC} - 23$		ns	
T_{CHWH}	CLKOUT High to $\overline{WR}$ Rising Edge	−10	10	ns	
T_{WLWH}	$\overline{WR}$ Low Period	$T_{OSC} - 30$		ns	
T_{WHQX}	Data Hold after $\overline{WR}$ Rising Edge	$T_{OSC} - 10$		ns	
T_{WHLH}	$\overline{WR}$ Rising Edge to ALE Rising Edge	$T_{OSC} - 10$	$T_{OSC} + 15$	ns	(Note 2)
T_{WHBX}	$\overline{BHE}$, INST HOLD after $\overline{WR}$ Rising Edge	$T_{OSC} - 10$		ns	

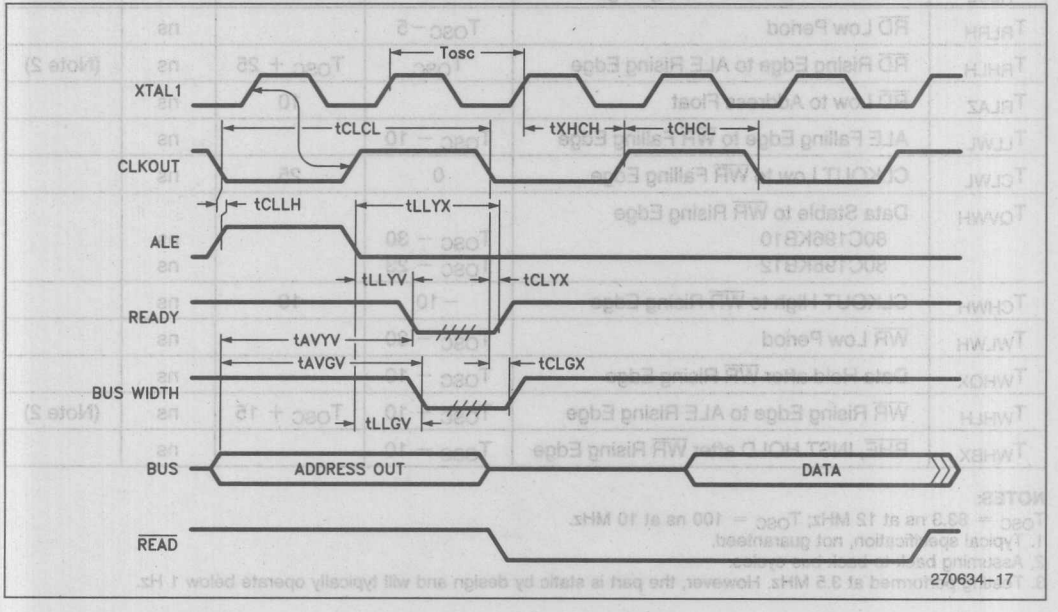
NOTES:

T_{OSC} = 83.3 ns at 12 MHz; T_{OSC} = 100 ns at 10 MHz.

1. Typical specification, not guaranteed.
2. Assuming back-to-back bus cycles.
3. Testing performed at 3.5 MHz. However, the part is static by design and will typically operate below 1 Hz.

System Bus Timings

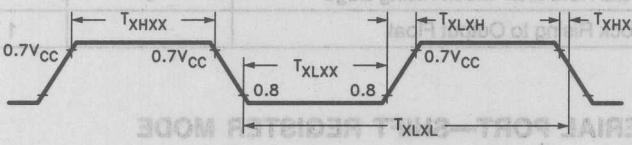




EXTERNAL CLOCK DRIVE

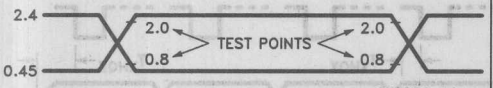
Symbol	Parameter	Min	Max	Units
$1/T_{XLXL}$	Oscillator Frequency			
	80C196KB10	3.5	10.0	MHz
	80C196KB12	3.5	12.0	MHz
T_{XLXL}	Oscillator Frequency			
	80C196KB10	100	286	ns
	80C196KB12	83	286	ns
T_{XHXX}	High Time	32		ns
T_{XLXX}	Low Time	32		ns
T_{XLXH}	Rise Time		10	ns
T_{XHXL}	Fall Time		10	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270634-18

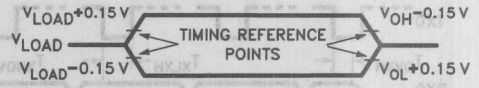
A.C. TESTING INPUT, OUTPUT WAVEFORM



270634-19

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0" Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



270634-20

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} = \pm 15$ mA.

EXPLANATION OF AC SYMBOLS

Each symbol is two pairs of letters prefixed by "T" for time. The characters in a pair indicate a signal and its condition, respectively. Symbols represent the time between the two signal/condition points.

Conditions:

H - High
L - Low
V - Valid
X - No Longer Valid
Z - Floating

Signals:

A - Address
B - $\overline{BHE}$
C - CLKOUT
D - DATA
G - Buswidth
L - ALE/$\overline{ADV}$
R - $\overline{RD}$
W - $\overline{WR}$/$\overline{WRH}$/$\overline{WRL}$
X - XTAL1
Y - READY

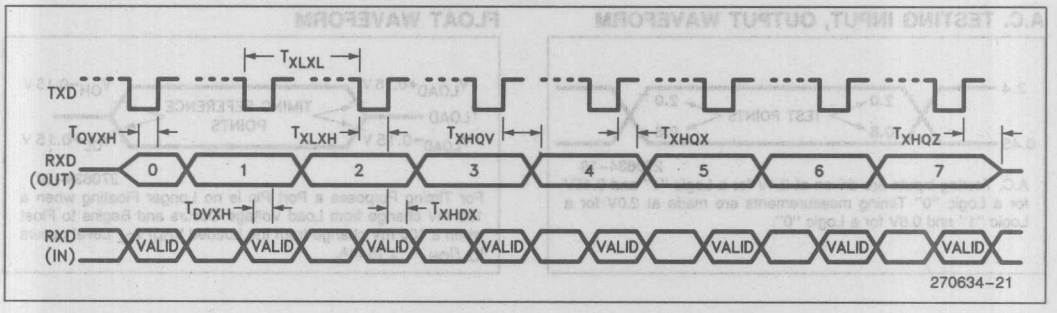
A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT TIMING—SHIFT REGISTER MODE (NOTE 1)

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period (BRR ≥ 8002H)	6 T_{OSC}		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge (BRR ≥ 8002H)	4 T_{OSC} ±50		ns
T_{XLXL}	Serial Port Clock Period (BRR = 8001H)	4 T_{OSC}		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge (BRR = 8001H)	2 T_{OSC} ±50		ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	2 T_{OSC} −50		ns
T_{XHQX}	Output Data Hold after Clock Rising Edge	2 T_{OSC} −50		ns
T_{XHQV}	Next Output Data Valid after Clock Rising Edge		2 T_{OSC} +50	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	T_{OSC} +50		ns
T_{XHDX}	Input Data Hold after Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		1 T_{OSC}	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



NOTE:
1. Timings are not tested but guaranteed by design.

A TO D CHARACTERISTICS

There are two modes of A/D operation: with or without clock prescaler. The speed of the A/D converter can be adjusted by setting a clock prescaler on or off. At high frequencies more time is needed for the comparator to settle. The maximum frequency with the clock prescaler disabled is 8 MHz. The conversion times with the prescaler turned on or off is shown in the table below.

The converter is ratiometric, so the absolute accuracy is directly dependent on the accuracy and stability of V_{REF}. V_{REF} must be close to V_{CC} since it supplies both the resistor ladder and the digital section of the converter.

Clock Prescaler On IOC2.4 = 0	Clock Prescaler Off IOC2.4 = 1
Mode 0–158 States 26.33 μs @ 12 MHz	Mode 2–91 States 22.75 μs @ 8 MHz

A/D CONVERTER SPECIFICATIONS

The specifications given below assume adherence to the Operating Conditions section of this data sheet. Testing is performed in Mode 2 with V_{REF} = 5.12V and 10 MHz on XTAL1.

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		256	1024 10	Levels Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.009 0.009 0.009			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation	−60			dB	1, 2, 3
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Time Slow Mode Fast Mode	15 8			States States	4 4
Input Capacitance	3			pF	

NOTES:
* These values are expected for most parts at 25°C but are not tested or guaranteed.
**An "LSB", as used here, has a value of approximately 5 mV.
1. These values are not tested in production and are guaranteed based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. Multiplexer Break-Before-Make Guaranteed.
4. One state = 167 ns at 12 MHz, 250 ns at 8 MHz.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected (e.g., the converter will not short inputs together).

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For an 8-bit converter with a reference voltage of 5.12V, one LSB is 20 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 40 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 80 mV.)

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE TIME—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

80C196KB FUNCTIONAL DEVIATIONS

The 80C196KB has the following problems.

1. The DJNZW instruction is not guaranteed to be functional. The instruction, if encountered, will not cause an unimplemented opcode. The DJNZ (byte instruction) works correctly and should be used instead.
2. The serial port only tolerates a +1.25%, −7.5% baud rate error between Transmitter and Receiver. If the serial port fails on the receiver, increase the baud rate.
3. The CDE function is not guaranteed to work. To ensure proper 80C196KB operation, the pin must be grounded.

DIFFERENCES BETWEEN THE 80C196KA AND THE 80C196KB

The 8XC196KB is identical to 8XC196KA except for the following differences.

1. ALE is high after reset on the 80C196KB instead of low as on the 80C196KA.
2. The DJNZW instruction is not guaranteed to work on the 80C196KB.
3. The $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ bus protocol is available on the 80C196KB.
4. Specification differences are outlined in the revision history section.

CONVERTING FROM OTHER 8096 FAMILY PRODUCTS TO THE 80C196KB

The following list of suggestions for designing an 809XBH system will yield a design that is easily converted to the 80C196KB.

1. Do not base critical timing loops on instruction or peripheral execution times.
2. Use equate statements to set all timing parameters, including the baud rate.
3. Do not base hardware timings on CLKOUT or XTAL1. The timings of the 80C196KB are different than those of the 8X9XBH, but they will function with standard ROM/EPROM/Peripheral type memory systems.
4. Make sure all inputs are tied high or low and not left floating.
5. Indexed and indirect operations relative to the stack pointer (SP) work differently on the 80C196KB than on the 8096. On the 8096, the address is calculated based on the un-updated version of the stack pointer. The 80C196KB uses the updated version. The offset for PUSH[SP], POP[SP], PUSH nn[SP] and POP nn[SP] instructions may need to be changed by a count of 2.

REVISION HISTORY

The 80C196KB is an improved version of the 80C196KA implementing the HOLD/HLDA bus protocol. The following differences exist between the 80C196KB and 80C196KA data sheets.

1. The no sample and hold feature on the A/D is no longer available.
2. The I_{IL1} Logical 0 input current in reset changed from $-500\ \mu A$ to $-850\ \mu A$.
3. The following A.C. Characteristics have changed on the 80C196KB.

Symbol	Parameter	80C196KA12	80C196KB12
T_{LLYV}	ALE Low to READY Setup	$T_{OSC} - 60$	$T_{OSC} - 72$
T_{LLGV}	ALE Low to Buswidth Setup	$T_{OSC} - 60$	$T_{OSC} - 70$
T_{LLAX}	Address Hold after ALE Falling	$T_{OSC} - 25$	$T_{OSC} - 40$
T_{LLRL}	ALE Falling Edge to RD Falling	$T_{OSC} - 30$	$T_{OSC} - 40$
T_{RLCL}	RD Low to CLKOUT Falling	5/25	10/30
T_{LLCH}	ALE Low to CLKOUT High	−10/10	−15/15
T_{WHBX}	$\overline{BHE}$, INST − HOLD after $\overline{WR}$ Rising Edge	$T_{OSC} - 5$	$T_{OSC} - 10$
T_{AVGV}	Address Valid to Buswidth Setup	$2T_{OSC} - 70$	$2T_{OSC} - 85$
T_{AVYV}	Address Valid to READY Setup	$2T_{OSC} - 70$	$2T_{OSC} - 85$
T_{AVDV}	Address Valid to Input Data Valid	$3T_{OSC} - 50$	$3T_{OSC} - 67$
T_{QVWH}	Data Stable to $\overline{WR}$ Rising Edge	$T_{OSC} - 20$	$T_{OSC} - 23$

4. T_{RLAZ} (Read Low to Address Float) was added as a specification (T_{RLAZ} = 10 ns).
5. New current specifications have been added to reflect typical current consumption at room temperature.
6. A graph of current consumption, both max and typical, over frequency for I_{CC} and I_{IDLE} has been added.
7. Sample time and input capacitance specifications for the A/D have been added.
8. The minimum instruction times for some of the indirect instructions which write to external memory have changed.
9. The LCC pinout description has been removed.
10. All functional deviations for the 80C196KA have been fixed except for the ones listed in the Functional Deviations sections.
11. Serial Port Timings have been added.
12. Some A/D Glossary Terms have been removed.
13. A/D sample time and sample capacitance have been added.
14. The NMI bit in IMASK1 should be set to zero for future compatibility.
15. The upper four bits of the A/D command should be set to zero for future compatibility.
16. HSO commands C and D should not be used to maintain future compatibility.

16-BIT HIGH PERFORMANCE CHMOS MICROCONTROLLER WITH 8 KBYTES OF ON-CHIP EPROM

- 232 Byte Register File
- Register-to-Register Architecture
- 28 Interrupt Sources/16 Vectors
- 2.3 μs 16 x 16 Multiply (12 MHz)
- 4.0 μs 32/16 Divide (12 MHz)
- Powerdown and Idle Modes
- Five 8-Bit I/O Ports
- 16-Bit Watchdog Timer
- Dynamically Configurable 8-Bit or 16-Bit Buswidth
- Full Duplex Serial Port
- High Speed I/O Subsystem
- 16-Bit Timer
- 16-Bit Up/Down Counter with Capture
- Pulse-Width-Modulated Output
- Four 16-Bit Software Timers
- 10-Bit A/D Converter with S/H
- $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ Bus Protocol
- 12 MHz Version—87C196KB12
  10 MHz Version—87C196KB10

The 87C196KB is an 80C196KB 16-bit microcontroller with 8 Kbytes of on-chip EPROM. Both parts are high performance members of the 8096 microcontroller family. The 87C196KB is pin-for-pin compatible and uses a true superset of the 8096 instructions. Intel's CHMOS process provides a high performance processor along with low power consumption. To further reduce power requirements, the processor can be placed into Idle or Powerdown Mode.

Bit, byte, word and some 32-bit operations are available on the 87C196KB. With a 12 MHz oscillator a 16-bit addition takes 0.66 μs, and the instruction times average 0.5 μs to 1.5 μs in typical applications.

Four high-speed capture inputs are provided to record times when events occur. Six high-speed outputs are available for pulse or waveform generation. The high-speed output can also generate four software timers or start an A/D conversion. Events can be based on the timer or up/down counter.

Also provided on-chip are an A/D converter, serial port, watchdog timer, and a pulse-width-modulated output signal.

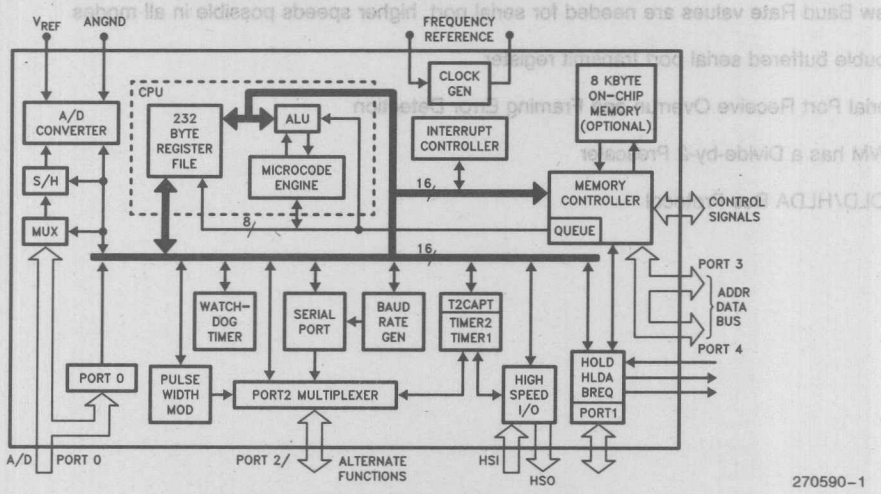


Figure 1. 87C196KB Block Diagram

October 1988
Order Number: 270590-002

ARCHITECTURE

The 87C196KB is a member of the MCS®-96 family, and as such has the same architecture and uses the same instruction set as the 8096. Many new features have been added on the 87C196KB including:

CPU FEATURES

Divide by 2 instead of divide by 3 clock for 1.5X performance

Faster instructions, especially indexed/indirect data operations

2.33 μs 16 × 16 multiply with 12 MHz clock (was 6.25 μs) on the 8096

Faster interrupt response (almost twice as fast as 8096)

Powerdown and Idle Modes

Clock Failure Detect

6 new instructions including Compare Long and Block Move

8 new interrupt vectors/6 new interrupt sources

PERIPHERAL FEATURES

SFR Window switching allows read-only registers to be written and vice-versa

Timer2 can count up or down by external selection

Timer2 has an independent capture register

HSO line events are stored in a register

HSO has CAM Lock and CAM Clear commands

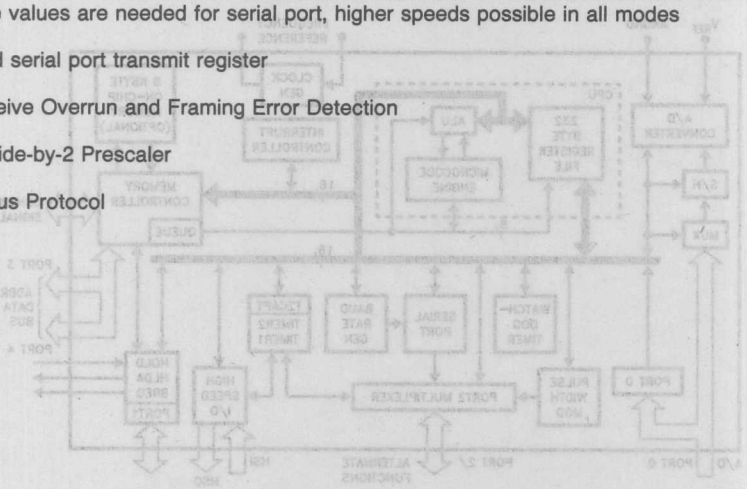
New Baud Rate values are needed for serial port, higher speeds possible in all modes

Double buffered serial port transmit register

Serial Port Receive Overrun and Framing Error Detection

PWM has a Divide-by-2 Prescaler

HOLD/HLDA Bus Protocol



NEW INSTRUCTIONS

PUSHA — PUSHes the PSW, IMASK, IMASK1, and WSR
(Used instead of PUSHF when new interrupts and registers are used.)

assembly language format: PUSHA
object code format: <11110100>
bytes: 1
states: on-chip stack: 12
off-chip stack: 18

POPA — POPs the PSW, IMASK, IMASK1, and WSR
(Used instead of POPF when new interrupts and registers are used.)

assembly language format: POPA
object code format: <11110101>
bytes: 1
states: on-chip stack: 12
off-chip stack:18

IDLPD — Sets the part into Idle or Powerdown Mode

assembly language format: IDLPD #key (key=1 for Idle, key=2 for Powerdown.)
object code format: <11110110> <key>
bytes: 2
states: legal key: 8
illegal key: 25

DJNZW* — Decrement Jump Not Zero using a Word counter

assembly language format: DJNZW wreg, cadd
object code format: <11100001> <wreg> <disp>
bytes: 3
states: jump not taken: 6
jump taken: 10

CMPL — Compare 2 long direct values

assembly language format:

	DST	SRC
CMPL	Lreg,	Lreg

object code format: <11000101> <src Lreg> <dst Lreg>
bytes: 3
states: 7

BMOV — Block move using 2 auto-incrementing pointers and a counter

assembly language format:

	PTRS	CNTREG
BMOV	Lreg,	wreg

object code format: <11000001> <wreg> <Lreg>
bytes: 3
states: internal/internal: 8 per transfer + 6
external/internal: 11 per transfer + 6
external/external: 14 per transfer + 6

*The DJNZW instruction is not guaranteed to work on this version of the 80C196KB. See the Functional Deviations section.

SFR OPERATION

All of the registers that were present on the 8096 work the same way as they did, except that the baud rate value is different. The new registers shown in the memory map control new functions. The most important new register is the Window Select Register (WSR) which allows reading of the formerly write-only registers and vice-versa. Using the WSR is described later in this data sheet.

PACKAGING

The 87C196KB is available in a 68-pin LCC package. Contact your local sales office to determine the exact ordering code for the part desired.

LCC	Description	LCC	Description	LCC	Description
1	ACH7/P0.7/PMD3	24	AD6/P3.6	47	P1.6/$\overline{\text{HLDA}}$
2	ACH6/P0.6/PMD2	25	AD7/P3.7	48	P1.5/$\overline{\text{BREQ}}$
3	ACH2/P0.2	26	AD8/P4.0	49	HSO.1
4	ACH0/P0.0	27	AD9/P4.1	50	HSO.0
5	ACH1/P0.1	28	AD10/P4.2	51	HSO.5/HSI.3/SID3
6	ACH3/P0.3	29	AD11/P4.3	52	HSO.4/HSI.2/SID2
7	NMI	30	AD12/P4.4	53	HSI.1/SID1
8	$\overline{\text{EA}}$	31	AD13/P4.5	54	HSI.0/SID0
9	V_{CC}	32	AD14/P4.6	55	P1.4
10	V_{SS}	33	AD15/P4.7	56	P1.3
11	XTAL1	34	T2CLK/P2.3	57	P1.2
12	XTAL2	35	READY	58	P1.1
13	CLKOUT	36	T2RST/P2.4/$\overline{\text{AINC}}$	59	P1.0
14	BUSWIDTH	37	$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	60	TXD/P2.0/PVER
15	INST	38	$\overline{\text{WR}}$/$\overline{\text{WRL}}$	61	RXD/P2.1/$\overline{\text{PALE}}$
16	ALE/$\overline{\text{ADV}}$	39	PWM/P2.5	62	$\overline{\text{RESET}}$
17	$\overline{\text{RD}}$	40	P2.7/T2CAPTURE/$\overline{\text{PACT}}$	63	EXTINT/P2.2/$\overline{\text{PROG}}$
18	AD0/P3.0	41	V_{PP}	64	CDE(1)
19	AD1/P3.1	42	V_{SS}	65	V_{REF}
20	AD2/P3.2	43	HSO.3	66	ANGND
21	AD3/P3.3	44	HSO.2	67	ACH4/P0.4/PMD0
22	AD4/P3.4	45	P2.6/T2UP-DN	68	ACH5/P0.5/PMD1
23	AD5/P3.5	46	P1.7/$\overline{\text{HOLD}}$		

Figure 2. Pin Definitions

NOTE:
1. The CDE function is not guaranteed to work. To ensure proper 87C196KB operation, the pin must be grounded.

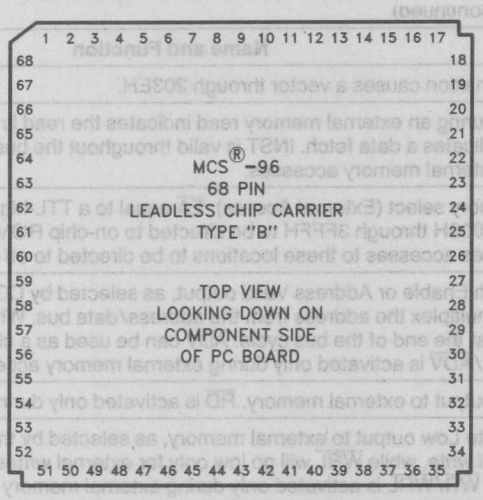


Figure 3. 68-Pin Package (LCC—Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
CDE(1)	Clock Detect Enable - When pulled high enables the clock failure detection circuit. If the XTAL1 frequency falls below a specified limit the $\overline{\text{RESET}}$ pin will be pulled low.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Timing pin for the return from powerdown circuit. Connect this pin with a 1 μF capacitor to V_{SS} and a 1 MΩ resistor to V_{CC}. If this function is not used V_{PP} may be tied to V_{CC}. This pin was V_{BB} on the 8X9X-90 parts and is the programming voltage.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ½ the oscillator frequency. It has a 50% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 4 state times to reset the chip. The subsequent low-to-high transition re- synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
NMI	A positive transition causes a vector through 203EH.
INST	Output high during an external memory read indicates the read is an instruction fetch and output low indicates a data fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.
$\overline{WR}$/$\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}$/$\overline{WRL}$ is activated only during external memory writes.
$\overline{BHE}$/$\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE} = 0$ selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{BHE}$/$\overline{WRH}$ is valid only during 16-bit external memory write cycles.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle (held not ready) is available through configuration of CCR.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as the SID in Slave Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. Three pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins set the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port. These pins are shared with $\overline{HOLD}$, $\overline{HLDA}$ and $\overline{BREQ}$.
Port 2	8-bit multi-functional port. All of its pins are shared with other functions in the 87C196KB.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups.
$\overline{HOLD}$	Bus Hold input requesting control of the bus. Enabled by setting WSR.7.
$\overline{HLDA}$	Bus Hold acknowledge output indicating release of the bus. Enabled by setting WSR.7.
$\overline{BREQ}$	Bus Request output activated when the bus controller has a pending external memory cycle. Enabled by setting WSR.7.

NOTE:
1. The CDE function is not guaranteed to work. To ensure proper 87C196KB operation, the pin must be grounded.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
TxD	The TxD pin is used for serial port transmission and reception in mode 0 only. The TxD function is enabled by setting IOC1.S. In mode 0 the pin is used as the serial clock output.
RxD	Serial Port Receive pin used for serial port reception. The RxD function is enabled by setting SPLON.3. In mode 0 the pin functions as input or output data.
EXTINT	A rising edge on the EXTINT pin will generate an external interrupt. EXTINT is selected as the external interrupt source by setting IOC1.1 high.
T2CLK	The T2CLK pin is the Timer2 clock input or the serial port baud rate generator input.
T2RST	A rising edge on the T2RST pin will reset Timer2. The T2RST function is enabled by setting IOCO.3. T2RST is enabled as the reset source by setting IOCO.5.
PWM	Port 2.5 can be enabled as a PWM output by setting IOC1.0. The duty cycle of the PWM is determined by the value loaded into the PWM-CONTROL register (17H).
T2UPDN	The T2UPDN pin controls the direction of Timer2 as an up or down counter. The Timer2 up/down function is enabled by setting IOC2.1.
T2CAP	A rising edge on (P2.7 will capture the value of Timer2 in the T2CAPTURE register (location 0CH in Window 15).

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	–	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	–	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	–	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	–	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	–	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	↕	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	↕	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	↕	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	↕	↑	–	
AND/ANDB	2	D ← D AND A	↕	↕	0	0	–	–	
AND/ANDB	3	D ← B AND A	↕	↕	0	0	–	–	
OR/ORB	2	D ← D OR A	↕	↕	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	↕	↕	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP − 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC ← (SP); SP ← SP + 2	–	–	–	–	–	–	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	–	–	–	–	–	–	5
JC	1	Jump if C = 1	–	–	–	–	–	–	5
JNC	1	jump if C = 0	–	–	–	–	–	–	5
JE	1	jump if Z = 1	–	–	–	–	–	–	5
JNE	1	Jump if Z = 0	–	–	–	–	–	–	5
JGE	1	Jump if N = 0	–	–	–	–	–	–	5
JLT	1	Jump if N = 1	–	–	–	–	–	–	5
JGT	1	Jump if N = 0 and Z = 0	–	–	–	–	–	–	5
JLE	1	Jump if N = 1 or Z = 1	–	–	–	–	–	–	5
JH	1	Jump if C = 1 and Z = 0	–	–	–	–	–	–	5
JNH	1	Jump if C = 0 or Z = 1	–	–	–	–	–	–	5
JV	1	Jump if V = 0	–	–	–	–	–	–	5
JNV	1	Jump if V = 1	–	–	–	–	–	–	5
JVT	1	Jump if VT = 1; Clear VT	–	–	–	–	0	–	5
JNVT	1	Jump if VT = 0; Clear VT	–	–	–	–	0	–	5
JST	1	Jump if ST = 1	–	–	–	–	–	–	5
JNST	1	Jump if ST = 0	–	–	–	–	–	–	5
JBS	3	Jump if Specified Bit = 1	–	–	–	–	–	–	5,6
JBC	3	Jump if Specified Bit = 0	–	–	–	–	–	–	5,6
DJNZ/ DJNZW	1	D ← D − 1; If D ≠ 0 then PC ← PC + 8-bit offset	–	–	–	–	–	–	5 10
DEC/DECB	1	D ← D − 1	↓	↓	↓	↓	↑	–	
NEG/NEGB	1	D ← 0 − D	↓	↓	↓	↓	↑	–	
INC/INCB	1	D ← D + 1	↓	↓	↓	↓	↑	–	
EXT	1	D ← D; D + 2 ← Sign (D)	↓	↓	0	0	–	–	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↓	↓	0	0	–	–	3
NOT/NOTB	1	D ← Logical Not (D)	↓	↓	0	0	–	–	
CLR/CLRB	1	D ← 0	1	0	0	0	–	–	
SHL/SHLB/SHLL	2	C ← msb - - - - - lsb ← 0	↓	↓	↓	↓	↑	–	7
SHR/SHRB/SHRL	2	0 → msb - - - - - lsb → C	↓	↓	↓	0	–	↓	7
SHRA/SHRAB/SHRAL	2	msb → msb - - - - - lsb → C	↓	↓	↓	0	–	↓	7
SETC	0	C ← 1	–	–	1	–	–	–	
CLRC	0	C ← 0	–	–	0	–	–	–	

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	0	VT ← 0	–	–	–	–	0	–	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interupts (I ← 0)	–	–	–	–	–	–	
EI	0	Enable All Interupts (I ← 1)	–	–	–	–	–	–	
NOP	0	PC ← PC + 1	–	–	–	–	–	–	
SKIP	0	PC ← PC + 2	–	–	–	–	–	–	
NORML	2	Left shift till msb = 1; D ← shift count	✓	✓	0	–	–	–	7
TRAP	0	SP ← SP − 2; (SP) ← PC; PC ← (2010H)	–	–	–	–	–	–	9
PUSHA	1	SP ← SP-2; (SP) ← PSW; PSW ← 0000H; SP ← SP-2; (SP) ← IMASK1/WSR; IMASK1 ← 00H	0	0	0	0	0	0	
POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	✓	✓	✓	✓	✓	✓	
IDLPD	1	IDLE MODE IF KEY=1; POWERDOWN MODE IF KEY =2; CHIP RESET OTHERWISE	–	–	–	–	–	–	
CMPL	2	D-A	✓	✓	✓	✓	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

NOTES:

1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is done. Operands is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D,D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D,D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.
10. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

Instruction Execution State Times(1)

MNEMONIC	DIRECT	IMMED	INDIRECT		INDEXED	
			NORMAL*	A-INC*	SHORT*	LONG*
ADD (3-op)	5	6	7/10	8/11	7/10	8/11
SUB (3-op)	5	6	7/10	8/11	7/10	8/11
ADD (2-op)	4	5	6/8	7/9	6/8	7/9
SUB (2-op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3-op)	5	5	7/10	8/11	7/10	8/11
SUBB (3-op)	5	5	7/10	8/11	7/10	8/11
ADDB (2-op)	4	4	6/8	7/9	6/8	7/9
SUBB (2-op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
MUL (3-op)	16	17	18/21	19/22	19/22	20/23
MULU (3-op)	14	15	16/19	17/19	17/20	18/21
MUL (2-op)	16	17	18/21	19/22	19/22	20/23
MULU (2-op)	14	15	16/19	17/19	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3-op)	12	12	14/17	15/18	15/18	16/19
MULUB (3-op)	10	10	12/15	13/15	12/16	14/17
MULB (2-op)	12	12	14/17	15/18	15/18	16/19
MULUB (2-op)	10	10	12/15	13/15	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3-op)	5	6	7/10	8/11	7/10	8/11
AND (2-op)	4	5	6/8	7/9	6/8	7/9
OR (2-op)	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3-op)	5	5	7/10	8/11	7/10	8/11
ANDB (2-op)	4	4	6/8	7/9	6/8	7/9
ORB (2-op)	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD/LDB	4	5	5/8	6/8	6/9	7/10
ST/STB	4	5	5/8	6/9	6/9	7/10
LDBSE	4	4	5/8	6/8	6/9	7/10
LDBZE	4	4	5/8	6/8	6/9	7/10
BMOV	6+8 per word			6 + 11/14 per word		
PUSH (int stack)	6	7	9/12	10/13	10/13	11/14
POP (int stack)	8	–	10/12	11/13	11/13	12/14
PUSH (ext stack)	8	9	11/14	12/15	12/15	13/16
POP (ext stack)	11	–	13/15	14/16	14/16	15/17

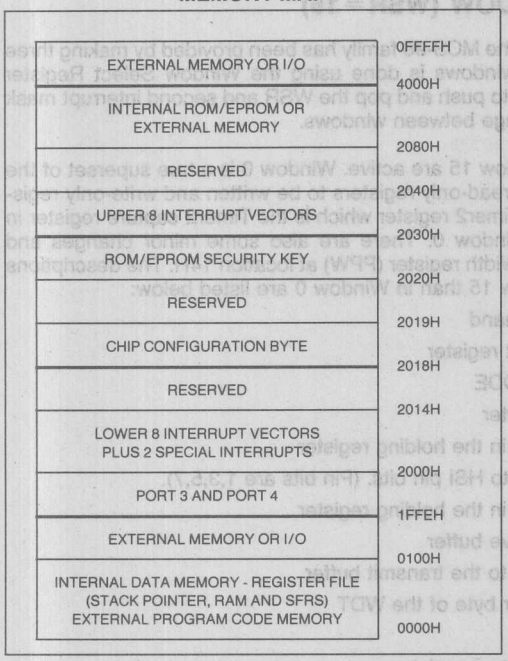
*Times for (Internal/External) Operands

NOTE:
1. Execution times for instructions accessing external data memory may be one to two states higher depending on the instruction stream being executed. In sixteen bit mode, the minimum execution state times apply for instruction accessing internal register space. Execution times do not reflect eight bit mode or insertion of wait states.

MNEMONIC		MNEMONIC	
PUSHF (int stack)	6	PUSHF (ext stack)	8
POPF (int stack)	7	POPF (ext stack)	10
PUSHA (int stack)	12	PUSHA (ext stack)	18
POPA (int stack)	12	POPA (ext stack)	18
TRAP (int stack)	16	TRAP (ext stack)	18
LCALL (int stack)	11	LCALL (ext stack)	13
SCALL (int stack)	11	SCALL (ext stack)	13
RET (int stack)	11	RET (ext stack)	14
CMPL	7	DEC/DECB	3
CLR/CLRB	3	EXT/EXTB	4
NOT/NOTB	3	INC/INCB	3
NEG/NEGB	3		
LJMP	7		
SJMP	7		
BR [indirect]	7		
JNST, JST	4/8 jump not taken/jump taken		
JNH, JH	4/8 jump not taken/jump taken		
JGT, JLE	4/8 jump not taken/jump taken		
JNC, JC	4/8 jump not taken/jump taken		
JNVT, JVT	4/8 jump not taken/jump taken		
JNV, JV	4/8 jump not taken/jump taken		
JGE, JLT	4/8 jump not taken/jump taken		
JNE, JE	4/8 jump not taken/jump taken		
JBC, JBS	5/9 jump not taken/jump taken		
DJNZ	5/9 jump not taken/jump taken		
DJNZW (Note 1)	5/9 jump not taken/jump taken		
NORML	8 + 1 per shift (9 for 0 shift)		
SHRL	7+ 1 per shift (8 for 0 shift)		
SHLL	7 + 1 per shift (8 for 0 shift)		
SHRAL	7 + 1 per shift (8 for 0 shift)		
SHR/SHRB	6 + 1 per shift (7 for 0 shift)		
SHL/SHLB	6 + 1 per shift (7 for 0 shift)		
SHRA/SHRAB	6 + 1 per shift (7 for 0 shift)		
CLRC	2		
SETC	2		
DI	2		
EI	2		
CLRVT	2		
NOP	2		
RST	15 (includes fetch of configuration byte)		
SKIP	3		
IDLPD	8/25 (proper key/improper key)		

NOTE:
1. The DJNZW instruction is not guaranteed to work. See Functional Deviations section.

MEMORY MAP



87C196KB INTERRUPTS

Number	Source	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT Pin	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

	WHEN READ	WHEN WRITTEN
19H	STACK POINTER	STACK POINTER
18H		
17H	*IOS2	PWM_CONTROL
16H	IOS1	IOC1
15H	IOS0	IOC0
14H	*WSR	*WSR
13H	*INT_MASK1	*INT_MASK1
12H	*INT_PEND1	*INT_PEND1
11H	*SP_STAT	*SP_CON
10H	PORT2	PORT2
0FH	PORT1	PORT1
0EH	PORT0	BAUD RATE
0DH	TIMER2 (HI)	TIMER2 (HI)
0CH	TIMER2 (LO)	TIMER2 (LO)
0BH	TIMER1 (HI)	*IOC2
0AH	TIMER1 (LO)	WATCHDOG
09H	INT_PEND	INT_PEND
08H	INT_MASK	INT_MASK
07H	SBUF(RX)	SBUF(TX)
06H	HSI_STATUS	HSO_COMMAND
05H	HSI_TIME (HI)	HSO_TIME (HI)
04H	HSI_TIME (LO)	HSO_TIME (LO)
03H	AD_RESULT (HI)	HSI_MODE
02H	AD_RESULT (LO)	AD_COMMAND
01H	ZERO REG (HI)	ZERO REG (HI)
00H	ZERO REG (LO)	ZERO REG (LO)

WSR = 0

	WSR = 15
0FH	RESERVED(1)
0EH	RESERVED(1)
0DH	*T2 CAPTURE (HI)
0CH	*T2 CAPTURE (LO)

OTHER SFRS IN WSR 15 BECOME READABLE:
IF THEY WERE WRITABLE IN WSR = 0 AND WRITABLE IF THEY WERE READABLE IN WSR = 0

	WSR = 14
4H	PPW

*NEW OR CHANGED REGISTER FUNCTION FROM 8096

NOTE:
1. RESERVED REGISTERS SHOULD NOT BE WRITTEN

USING THE ALTERNATE REGISTER WINDOW (WSR = 15)

I/O register expansion on the new CHMOS members of the MCS-96 family has been provided by making three register windows available. Switching between these windows is done using the Window Select Register (WSR). The PUSHA and POPA instructions can be used to push and pop the WSR and second interrupt mask when entering or leaving interrupts, so it is easy to change between windows.

On the 87C196KB only Window 0, Window 14 and Window 15 are active. Window 0 is a true superset of the standard 8096 SFR space, while Window 15 allows the read-only registers to be written and write-only registers to be read. The only major exception to this is the Timer2 register which is the Timer2 capture register in Window 15. The writeable register for Timer2 is in Window 0. There are also some minor changes and cautions. Window 14 contains the Programmable Pulse Width register (PPW) at location 14H. The descriptions of the registers which have different functions in Window 15 than in Window 0 are listed below:

AD_COMMAND (02H) — Read the last written command

AD_RESULT (02H, 03H) — Write a value into the result register

HSI_MODE (03H) — Read the value in HSI_MODE

HSI_TIME (04H,05H) — Write to FIFO Holding register

HSO_TIME (04H,05H) — Read the last value placed in the holding register

HSI_STATUS (06H) — Write to status bits but not to HSI pin bits. (Pin bits are 1,3,5,7).

HSO_COMMAND (06H) — Read the last value placed in the holding register

SBUF(RX) (07H) — Write a value into the receive buffer

SBUF(TX) (07H) — Read the last value written to the transmit buffer

WATCHDOG(0AH) — Read the value in the upper byte of the WDT

TIMER1 (0AH,0BH) — Write a value to Timer1

TIMER2 (0CH,0DH) — Read/Write the Timer2 capture register.
Note that Timer2 read/write is done with WSR = 0.

IOC2 (0BH) — Last written value is readable, except bit 7 (note 1)

BAUD_RATE (0EH) — No function, cannot be read

PORT0 (0EH) — No function, no output drivers on the pins. Registers reserved.

PORT1 (OFH) — IOPORT1 cannot be read or written in Window 15. Register reserved.

SP_STAT (11H) — Set the status bits, TI and RI can be set, but it will not cause an interrupt

SP_CON (11H) — Read the current control byte

IOS0 (15H) — Writing to this register controls the HSO pins. Bits 6 and 7 are inactive for writes.

IOC0 (15H) — Last written value is readable, except bit 1 (note 1)

IOS1 (16H) — Writing to this register will set the status bits, but not cause interrupts. Bits 6 and 7 are not functional

IOC1 (16H) — Last written value is readable

IOS2 (17H) — Writing to this register will set the status bits, but not cause interrupts.

PWM_CONTROL (17H) — Read the duty cycle value written to PWM_CONTROL

NOTE:

1. IOC2.7 (CAM CLEAR) and IOC0.1 (T2RST) are not latched and will read as a 1 (precharged bus) .

Being able to write to the read-only registers and vice-versa provides a lot of flexibility. One of the most useful advantages is the ability to set the timers and HSO lines for initial conditions other than zero.

Reserved registers may be used for testing or future features. Do not write to these registers. Reads from reserved registers will return indeterminate values.

SFR BIT SUMMARY

A summary of the SFRs which control I/O functions has been included in this section. The summary is separated into a list of those SFRs which have changed on the 87C196KB and a list of those which have remained the same.

The following 87C196KB SFRs are different than those on the 8096BH:

(The Read and Write comments indicate the register's function in Window 0 unless otherwise specified.)

SBUF(TX): Now double buffered
07h
write

BAUD RATE: Uses new Baud Rate Values
0Eh
write

SP_STAT:

7	6	5	4	3	2	1	0
RB8/ RPE	RI	TI	FE	TXE	OE	X	X

11h
read

RPE : Receive Parity Error
RI : Receive Indicator
TI : Transmit Indicator
FE : Framing Error
TXE : Transmitter Empty
OE : Receive Overrun Error

IPEND1:
IMASK1:

7	6	5	4	3	2	1	0
NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI

12h,13h
read/write

NMI : Non-Maskable Interrupt (Set to zero for future compatibility)
FIFO FULL : HSIO FIFO full
EXTINT : External Interrupt Pin
T2OVF : Timer2 Overflow
T2CAP : Timer2 Capture
HSI4 : HSI has 4 or more entries in FIFO
RI : Receive Interrupt
TI : Transmit Interrupt

WSR:

7	6	5	4	3	2	1	0
HLDEN	0	0	0	W	W	W	W

14h
read/write

WWWW = 0 : SFRs function like a superset of 8096 SFRs
WWWW = 14 : PPW register
WWWW = 15 : Exchange read/write registers
WWWW = OTHER : Undefined, do not use
000 : These bits must always be written as zeros to provide compatibility with future products.
HLDEN = 1 : Enables the HOLD/HLDA bus protocol

IOS2:

7	6	5	4	3	2	1	0
START A2D	T2 RESET	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

17h
read

Indicates which HSO event occured
START A2D : HSO_CMD 15, start A to D
T2RESET : HSO_CMD 14, Timer 2 reset
HSO.0-5 : Output pins HSO.0 through HSO.5

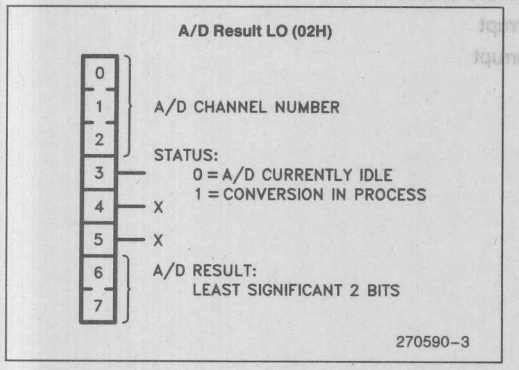
IOC2:

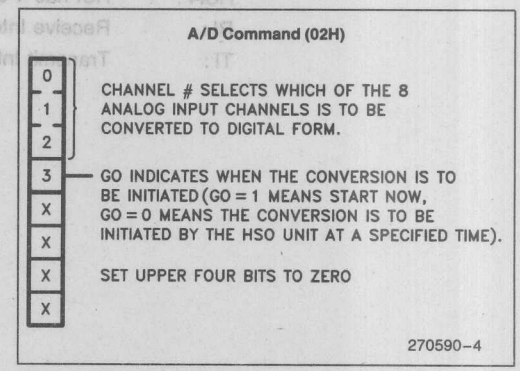
7	6	5	4	3	2	1	0
CLEAR CAM	ENA LOCK	T2ALT INT	A2D CPD	X	SLOW PWM	T2UD ENA	FAST T2EN

0Bh
write

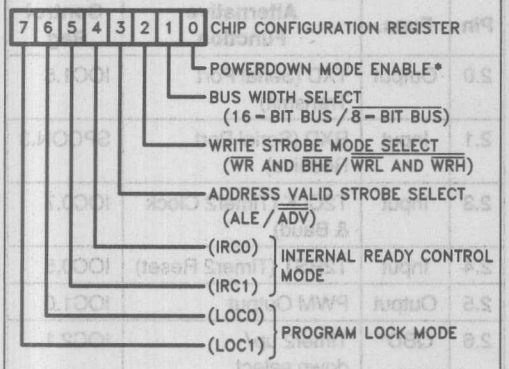
CLEAR_CAM : Clear Entire CAM
ENA_LOCK : Enable lockable CAM entry feature
T2ALT INT : Enable T2 Alternate Interrupt at 8000H
A2D_CPD : Clock Prescale Disable for low XTAL frequency (A to D conversion in fewer state times)
X : Set to zero
SLOW_PWM : Turn on divide by 2 Prescaler on PWM
T2UD ENA : Enable Timer 2 as up/down counter
FAST_T2EN : Enable Fast increment of T2; once per state time.

The following registers are the same on the 87C196KB as they were on the 8096BH:



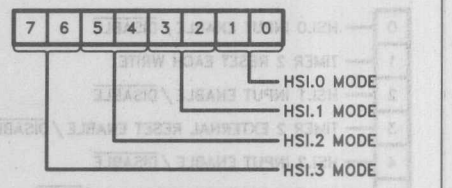


Chip Configuration (2018H)



*Minor Change

HSI_Mode (03H)

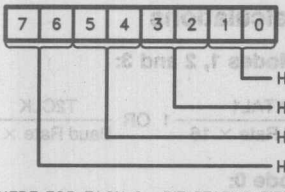


WHERE EACH 2 – BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

270590-6

HSI_Status (06H)

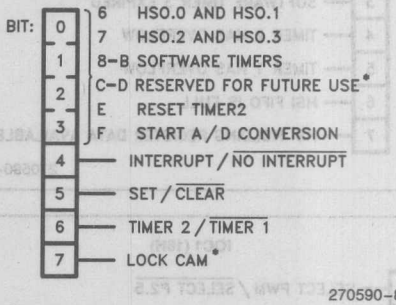


WHERE FOR EACH 2 – BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270590-7

HSO Command (06H)

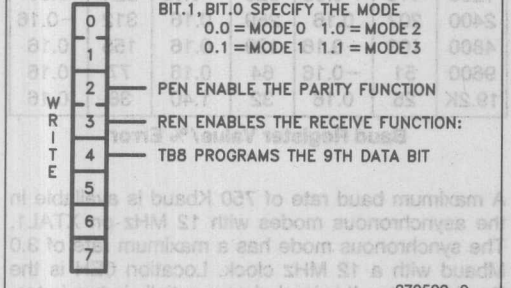
CHANNEL: 0–5 HS0.0 – HS0.5



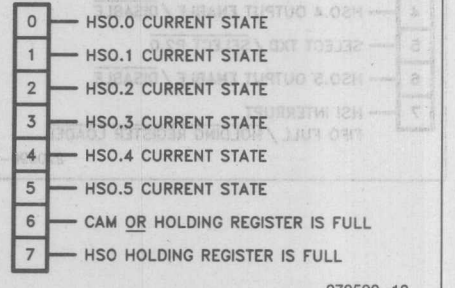
270590-8

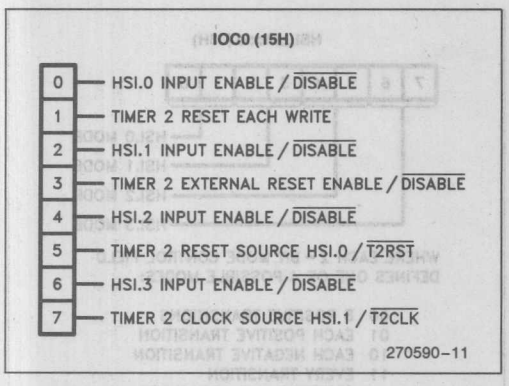
*Minor Change

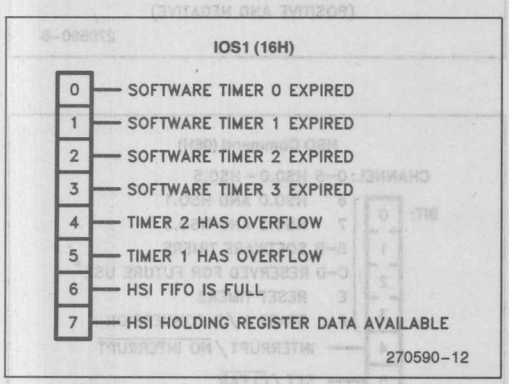
SPCON (11H)

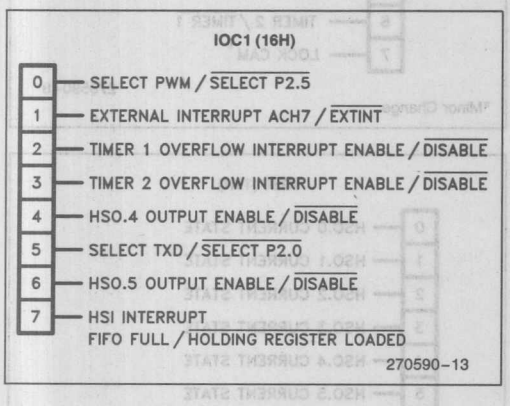


IOS0 (15H)









Port 2 Multiple Functions

Pin	Func.	Alternative Function	Control Reg.
2.0	Output	TXD (Serial Port Transmit)	IOC1.5
2.1	Input	RXD (Serial Port Receive)	SPCON.3
2.3	Input	T2CLK (Timer2 Clock & Baud)	IOC0.7
2.4	Input	T2RST (Timer2 Reset)	IOC0.5
2.5	Output	PWM Output	IOC1.0
2.6	QBD*	Timer2 up/ down select	IOC2.1
2.7	QBD*	Timer2 Capture	N/A

*QBD = Quasi-bidirectional

Baud Rate Calculations

Asynchronous Modes 1, 2 and 3:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 16} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate} \times 8}$$

Synchronous Mode 0:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 2} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate}}$$

Baud Rates and Baud Register Values

Baud Rate	XTAL Frequency					
	8.0 MHz		10.0 MHz		12.0 MHz	
300	1666	−0.02	2082	0.02	2499	0.00
1200	416	−0.08	520	−0.03	624	0.00
2400	207	0.16	259	0.16	312	−0.16
4800	103	−0.16	129	0.16	155	0.16
9600	51	−0.16	64	0.16	77	0.16
19.2K	25	0.16	32	1.40	38	0.16

Baud Register Value/% Error

A maximum baud rate of 750 Kbaud is available in the asynchronous modes with 12 MHz on XTAL1. The synchronous mode has a maximum rate of 3.0 Mbaud with a 12 MHz clock. Location 0EH is the Baud Register. It is loaded sequentially in two bytes, with the low byte being loaded first. This register may not be loaded with zero in serial port Mode 0.

NOTE:
The maximum T2CLK rate is 3 MHz when used to set the baud rate.

HOLD/HLDA PROTOCOL

The 87C196KB supports a bus exchange protocol, allowing other devices that are capable of acting as bus masters to gain control of the bus. The hand shake consists of three signals, $\overline{HOLD}$, $\overline{HLDA}$ and $\overline{BREQ}$. $\overline{HOLD}$, which is an input, is activated by a bus master which requires control of the bus. The 87C196KB responds by releasing the bus and activating the $\overline{HLDA}$ output line. When the new bus master is finished with the bus, it returns control of the bus to the 87C196KB by dropping its hold request. In response, the 87C196KB removes its hold acknowledge and assumes control of the bus. The third signal, $\overline{BREQ}$, is activated by the 87C196KB when it is in hold and has a pending external bus cycle. The 87C196KB deactivates this signal at the same time it removes the acknowledge signal. Figure 4 shows the bus timing for $\overline{HOLD}$/$\overline{HLDA}$.

The $\overline{HOLD}$, $\overline{HLDA}$ and $\overline{BREQ}$ signals are multiplexed with P1.7, P1.6 and P1.5 respectively. To enable the bus hold feature on these pins instead of the Port1 feature, the HLDEN(WSR.7) must be set to one. This bit is cleared during reset and must be set by software to configure the pins as well as to enable the bus hold feature. Once this bit is set, the three most significant pins of Port1 cannot be restored to their I/O function without resetting the part. The bus hold feature, however, can be disabled by clearing the HLDEN.

When the 87C196KB acknowledges the hold request, the output buffers for Port3, Port4 and all the bus control signals, (ALE, $\overline{BHE}$, INST, $\overline{RD}$ and $\overline{WR}$), are turned off. Although the strong pullup and pulldown on ALE are disabled during hold acknowledge, a weak pulldown is turned on to provide an option to wire OR the ALE with other bus masters. The request to hold acknowledge latency is dependent on the state of the bus controller.

MAXIMUM HOLD LATENCY

The maximum hold latency is the maximum time before the 87C196KB will respond with HLDA to an incoming HOLD. The minimum hold latency is half a state with one or two states added depending upon if the part is executing out of internal or external memory. The latency for entering the bus hold state from idle mode is the same as the timing when the part is executing out of the internal memory. The maximum latency is shown below:

Table 2. Maximum Hold Latency

	Max Hold Latency
Internal Execution	1½ States
External Execution	2½ States
Idle Mode	1½ States

REGAINING BUS CONTROL

There is no delay between the time that the 87C196KB removes the $\overline{HLDA}$ and when it takes control of the bus. After the hold request is removed, the 87C196KB will drop the hold acknowledge in the following state and assume control of the bus.

$\overline{BREQ}$ is activated when the part is in hold and it needs to do an external memory cycle. Once activated, it stays active until the hold request is removed. At the earliest, this signal can go active with $\overline{HLDA}$, regardless of which memory space the part was executing out of when the hold was requested.

Hold requests do not freeze the 87C196KB when executing out of internal memory. The part continues executing and servicing interrupt requests as long as the code and data needed reside in internal memory. As soon as the part needs to access external memory, it stops executing and waits for the hold request to be removed. At this point, the part cannot respond to any interrupt requests until hold request is dropped.

A hold request will freeze the 87C196KB if it is executing out of the external memory. After the 87C196KB acknowledges the request, it continues running until either the instruction queue becomes empty or it needs to perform an external data cycle. Normally, this will happen a few states after the 87C196KB grants $\overline{HLDA}$. In this situation the 87C196KB is not able to service any interrupt requests. The 87C196KB will respond to hold requests while in the idle mode. (In the idle mode, the values of the Port 3 and Port 4 data registers are forced onto the port pins and the control signals are driven.)

DISABLING HOLD REQUESTS

HLDEN(WSR.7) can be cleared to block hold requests in cases where consecutive memory cycles are required. Clearing this bit, however, does not cause the 87C196KB to take control of the bus immediately. The part waits until the current hold request is over, then it disables the bus hold feature, causing a new hold request to be ignored until HLDEN is set again. Since there is a delay from the time the code for clearing the bit is fetched to the time the bit is actually cleared, the code that clears HLDEN must be a few instructions ahead of the block that needs to be protected against hold requests. The safest way is to add a JNB instruction to check the status of the HLDA pin after the code that clears the HLDEN bit. A code example is given in Figure 5. This prevents the part from executing a new instruction until both current hold requests are serviced and the hold feature is disabled.

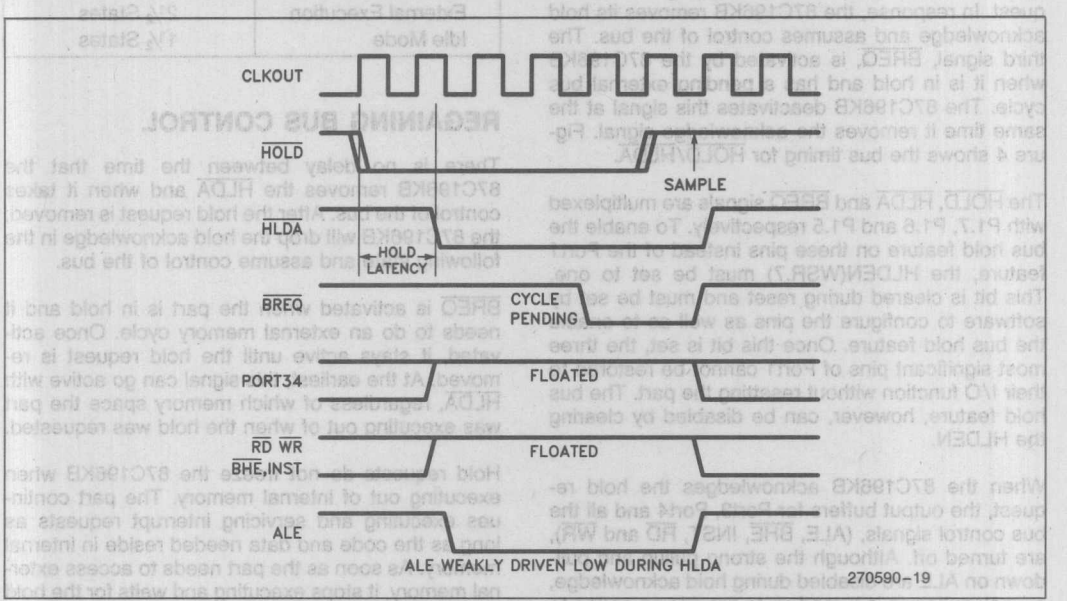


Figure 4. HOLD/HLDA Timing

```
        DI                  ; disable interrupts
        ANDB WSR, #0EFH     ; disable hold request
WAIT:   JBC PORT1, 6, WAIT  ; Check the HLDA pin
          .                 ; If set, execute
          .                 ; protected instructions
          .
        ORB WSR,#80h        ; enable HOLD requests
        EI                  ; enable interrupts
```

NOTE:
Interrupts should be disabled to prevent code interruption

Figure 5. Sample Code For Disabling HOLD/HLDA

EPROM CHARACTERISTICS

The 87C196KB contains 8 Kbytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when reset is asserted will place the 87C196KB in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 87C196KB is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

PROGRAMMING THE 87C196KB

Three flexible EPROM programming modes are available on the 87C196KB-auto, slave and run-time. These modes can be used to program 87C196KBs in a stand alone, gang or run-time environment.

The Auto Programming Mode enables an 87C196KB to program itself with the 8 Kbytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 87C196KBs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 7 shows how the pins are renamed. Figure 8 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 6). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 6. Programming Function PMODE Values

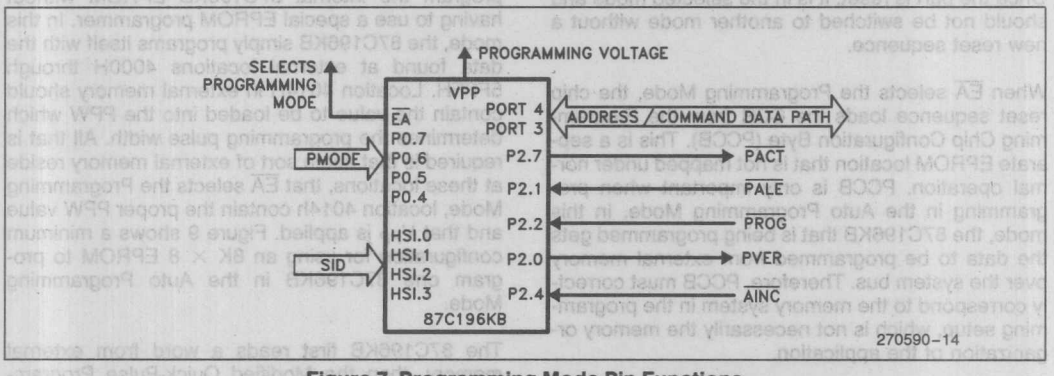


Figure 7. Programming Mode Pin Functions

Name	Function
PMODE	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 001 will use Port 3.1 to signal correct or incorrect program verification.
$\overline{\text{PALE}}$	Programming ALE Input. Accepted by an 87C196KB that is in Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming. Falling edge indicates valid data on PBUS and the beginning of programming. Rising edge indicates end of programming.
$\overline{\text{PACT}}$	Programming Active. Used in the Auto Programming Mode to indicate when programming activity is complete.
PVER	Program Verification. Signal is low after rising edge of PROG if the programming was not successful.
$\overline{\text{AINC}}$	Auto Increment. Active low signal indicates that the auto increment mode is enabled. Auto Increment will allow reading or writing of sequential EPROM locations without address transactions across the PBUS for each read or write.
PORTS	Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 87C196KBs. Used by chips in Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Should have pullups to V_{CC} (15 kΩ).

Figure 8. Programming Mode Pin Definitions

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{\text{RESET}}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{\text{EA}}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCB is only important when programming in the Auto Programming Mode. In this mode, the 87C196KB that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCB must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 87C196KB programming in each programming mode.

AUTO PROGRAMMING MODE

The Auto Programming Mode provides the ability to program the internal 87C196KB EPROM without having to use a special EPROM programmer. In this mode, the 87C196KB simply programs itself with the data found at external locations 4000H through 5FFFH. Location 4014H in external memory should contain the value to be loaded into the PPW which determines the programming pulse width. All that is required is that some sort of external memory reside at these locations, that $\overline{\text{EA}}$ selects the Programming Mode, location 4014h contain the proper PPW value and that V_{PP} is applied. Figure 9 shows a minimum configuration for using an 8K × 8 EPROM to program one 87C196KB in the Auto Programming Mode.

The 87C196KB first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be

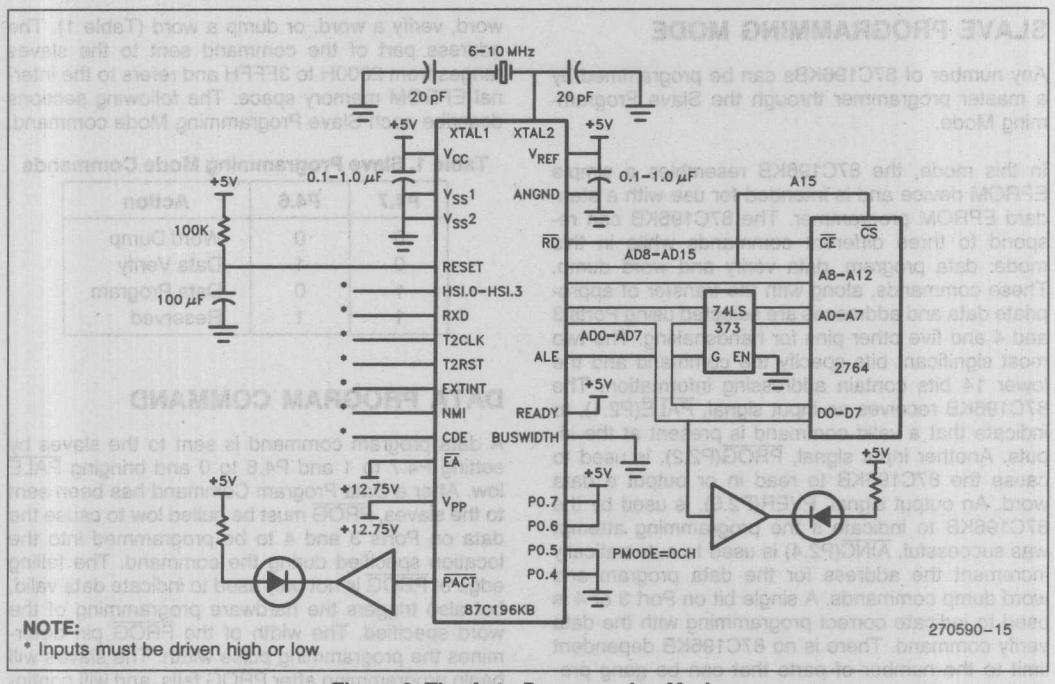


Figure 9. The Auto Programming Mode

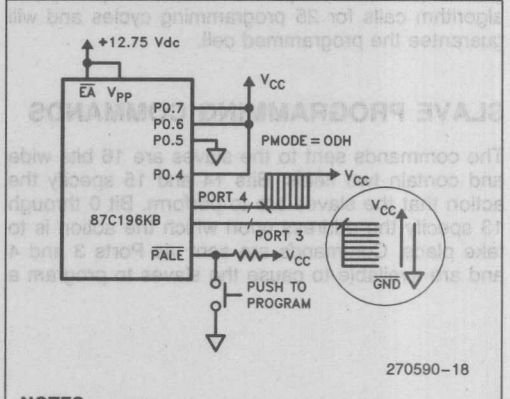
programmed is 0FFH. When all 8K have been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on Port 3.0 if it programmed correctly and a 1 if it failed.

AUTO CONFIGURATION BYTE PROGRAMMING MODE

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 10 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and 0FFh on Port 4, the CCB and PCCB will be programmed with the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the byte's programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on $\overline{EA}$ puts the 87C196KB in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 87C196KB is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).



NOTES:
Tie Port 3 to the value desired to be programmed into CCB and PCCB.
Make all necessary minimum connections for power, ground and clock.

Figure 10. The Auto CCB Programming Mode

SLAVE PROGRAMMING MODE

Any number of 87C196KBs can be programmed by a master programmer through the Slave Programming Mode.

In this mode, the 87C196KB resembles a simple EPROM device and is intended for use with a standard EPROM programmer. The 87C196KB can respond to three different commands while in this mode: data program, data verify and word dump. These commands, along with the transfer of appropriate data and addresses are selected using Ports 3 and 4 and five other pins for handshaking. The two most significant bits specify the command and the lower 14 bits contain addressing information. The 87C196KB receives an input signal, $\overline{\text{PALE}}$(P2.1), to indicate that a valid command is present at the inputs. Another input signal, $\overline{\text{PROG}}$(P2.2), is used to cause the 87C196KB to read in or output a data word. An output signal, PVER(P2.0), is used by the 87C196KB to indicate if the programming attempt was successful. $\overline{\text{AINC}}$(P2.4) is used to automatically increment the address for the data program and word dump commands. A single bit on Port 3 or 4 is used to indicate correct programming with the data verify command. There is no 87C196KB dependent limit to the number of parts that can be gang programmed in the slave mode.

In order to guarantee programming for the life of the EPROM, the modified quick pulse algorithm should be used. Programming the part using a simple programming pulse at V_{CC} and verified using PVER, will only guarantee the programmed cell at V_{CC}. The programmed cell will not be guaranteed at V_{CC} margins for the life of the part. The modified quick pulse algorithm calls for 25 programming cycles and will guarantee the programmed cell.

SLAVE PROGRAMMING COMMANDS

The commands sent to the slaves are 16 bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bit 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 1). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each Slave Programming Mode command.

Table 1. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND

A data program command is sent to the slaves by setting P4.7 to 1 and P4.6 to 0 and bringing $\overline{\text{PALE}}$ low. After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The width of the $\overline{\text{PROG}}$ pin determines the programming pulse width. The slaves will begin programming after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the data verify command. The $\overline{\text{AINC}}$ pin can be used to increment to the next location or a new command can be issued.

If PVER of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful for every slave. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. Figure 8 shows the relationship of $\overline{\text{PALE}}$, $\overline{\text{PROG}}$, and PVER to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

EPROM PROGRAMMING WAVEFORMS

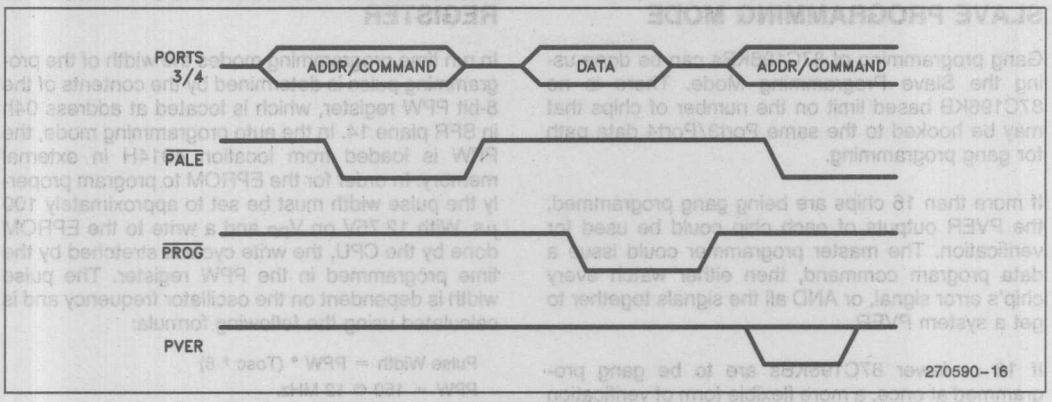


Figure 11. Data Program Signals in Slave Programming Mode

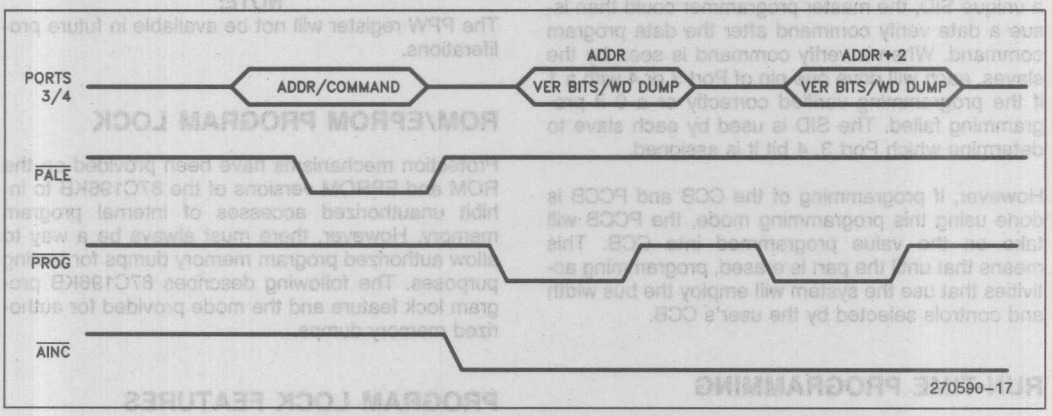


Figure 12. Data Verify Command Signals

DATA VERIFY COMMAND

When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 or 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. If the SID is 13 then the P4.5 pin is the indicator. $\overline{PROG}$ from the programmer governs when the slaves drive the bus. Figure 10 shows the relationship of Ports 3 and 4 to $\overline{PALE}$ and $\overline{PROG}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND

When the Word Dump Command is issued, the 87C196KB being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{PROG}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 11.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

GANG PROGRAMMING WITH THE SLAVE PROGRAMMING MODE

Gang programming of 87C196KBs can be done using the Slave Programming Mode. There is no 87C196KB based limit on the number of chips that may be hooked to the same Port3/Port4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER outputs of each chip could be used for verification. The master programmer could issue a data program command, then either watch every chip's error signal, or AND all the signals together to get a system PVER.

If 16 or fewer 87C196KBs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned.

However, if programming of the CCB and PCCB is done using this programming mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system will employ the bus width and controls selected by the user's CCB.

RUN-TIME PROGRAMMING

Run-Time Programming of the 87C196KB is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to V_{PP}. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation-internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. The value of the PPW register will determine the programming pulse. The PPW register may not be available on future versions of the MCS-96 family. To ensure future compatibility, an algorithm using the idle mode should be used to program the part. Figure 13 shows the algorithm which is recommended. The modified quick pulse algorithm is used to guarantee the programmed EPROM cell for the life of the part.

PROGRAMMING PULSE WIDTH REGISTER

In run time programming modes the width of the programming pulse is determined by the contents of the 8-bit PPW register, which is located at address 04h in SFR plane 14. In the auto programming mode, the PPW is loaded from location 4014H in external memory. In order for the EPROM to program properly the pulse width must be set to approximately 100 μs. With 12.75V on V_{PP} and a write to the EPROM done by the CPU, the write cycle is stretched by the time programmed in the PPW register. The pulse width is dependent on the oscillator frequency and is calculated using the following formula:

Pulse Width = PPW * (Tosc * 8)

PPW = 150 @ 12 MHz.

NOTE:
The PPW register will not be available in future proliferations.

ROM/EPROM PROGRAM LOCK

Protection mechanisms have been provided on the ROM and EPROM versions of the 87C196KB to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 87C196KB program lock feature and the mode provided for authorized memory dumps.

PROGRAM LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by setting the LOC0 bit in the CCR to 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable V_{PP} to the EPROM. This protects the entire EPROM 2000H-3FFFH from inadvertent or unauthorized programming.

Read protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H-3FFFH if the slave program counter is in the range 2000H-

```
                LD WSR,#14                      ;Initialize programmable
                LD PPW,#VALUE                   ;pulse width
PROGRAM:        POP ADDRESS__TEMP               ;Load program data
                POP DATA__TEMP                  ;and address
              * PUSHF
                LD COUNT, #25T                  ;program using Modified
                                                 Quick Pulse
LOOP:         * LDB INT__MASK,#ENABLE__SWT       ;program SWT for
              * LDB HSO__COMMAND,#SWTO__OVF      ;program pulse width
              * ADD HSO__TIME,TIMER1,#PROGRAM__PULSE
              * EI
                ST DATA__TEMP,[ADDR__TEMP]      ;enter idle mode until
              * IDLPD 1                          ;swt expires
                DJNZ COUNT, LOOP                ;loop 25 times
              * POPF
                RET
SWT__EXPIRED:   *POP 0                          ;service swt and return
              *RET
```

NOTE:
*Not Really Needed on Current 87C196KB Part

Figure 13. Future Run-Time Programming Algorithm

3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even though the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

The value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes, a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128-bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110b).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H-402FH the same 16-byte key that resides inside the chip at locations 2020H-202FH. Before doing a ROM dump, the chip checks that the keys match.

When programming using the auto configuration or auto programming modes, security key verification will be done if the PPCB has read and/or write protection enabled. After a successful key verification, the chip dumps data to external locations 4000H-5FFFH. Unspecified data appears at the low address. Internal EPROM/ROM is dumped to 4000H-5FFFH beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:
Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

The Modified Quick Pulse Algorithm

The Modified Quick-Pulse Programming Algorithm calls for each EPROM location to receive 25 separate 100 μs ($\pm$5 μs) programming cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 87C196KB parts is done with V_{PP} = 12.75V $\pm$ 0.25V and V_{CC} = 5.0V $\pm$ 0.5V.

The modified Quick Pulse Algorithm must be used to guarantee programming for the life of the EPROM in run-time and slave programming modes.

SIGNATURE WORD

The 87C196KB contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command. The programming voltage levels can be determined by reading locations 2072H and 2073h. The voltage levels are determined by the following equation.

Voltage = 20/256 * (test ROM data)

The stored values for the signature word and voltage levels are shown in Table 3.

Table 3. Signature Word and Voltage Levels

Description	Location	Value
Signature Word	2070H	897CH
Programming V_{CC}	2072H	040H (5.0V)
Programming V_{PP}	2073H	0A3H (12.75V)

ERASING THE 87C196KB

Initially, and after each erasure, all bits of the 87C196KB are in the "1" state. Data is introduced by selectively programming "0"s into the desired bit locations. Although only "0"s will be programmed, both "1"s and "0"s can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 87C196KB are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). If should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 87C196KB in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 87C196KB is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure. In power down, the part will draw more current than normal if the EPROM window is exposed to light.

The recommended erasure procedure for the 87C196KB is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (i.e., UV intensity $\times$ exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 87C196KB should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 87C196KB can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 87C196KB to high intensity UV light for long periods may cause permanent damage.

RESERVED LOCATION WARNING

Intel Reserved addresses cannot be used by applications which use 87C196KB internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be 0FFFFH to ensure a harmless result. A memory map indicating reserved locations on the 87C196KB is shown in Figure 14.

Intel Reserved locations, when mapped to external memory, must be filled with 0FFFFH to ensure compatibility with future parts.

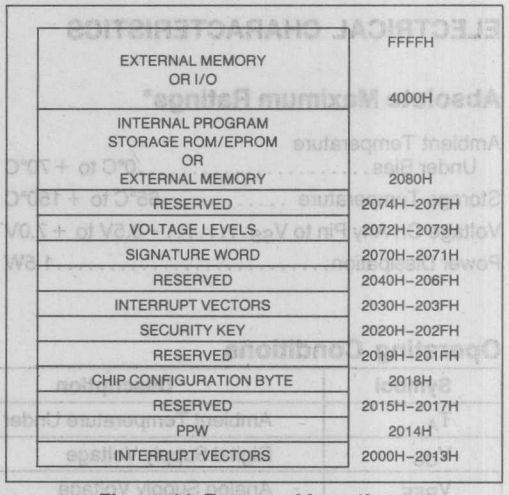


Figure 14. Reserved Locations

ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings*

Ambient Temperature Under Bias . . . . . 0°C to +70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage On Any Pin to V_{SS} . . . . . −0.5V to +7.0V
Power Dissipation 1.5W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

Operating Conditions

Symbol	Description	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	°C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	3.5	12	MHz

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. Characteristics (Over specified operating conditions)

Symbol	Description	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage (Note 1)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage on XTAL 1	0.7 V_{CC}	V_{CC} + 0.5	V	
V_{IH2}	Input High Voltage on RESET	2.2	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.3 0.45 1.5	V V V	I_{OL} = 200 µA I_{OL} = 3.2 mA I_{OL} = 7 mA
V_{OH}	Output High Voltage (Standard Outputs)	V_{CC} − 0.3 V_{CC} − 0.7 V_{CC} − 1.5		V V V	I_{OH} = −200 µA I_{OH} = −3.2 mA I_{OH} = −7 mA
V_{OH1}	Output High Voltage (Quasi-bidirectional Outputs)	V_{CC} − 0.3 V_{CC} − 0.7 V_{CC} − 1.5		V V V	I_{OH} = −10 µA I_{OH} = −30 µA I_{OH} = −60 µA
I_{LI}	Input Leakage Current (Std. Inputs)		±10	µA	$0 < V_{IN} < V_{CC} - 0.3V$
I_{LI1}	Input Leakage Current (Port 0)		±3	µA	$0 < V_{IN} < V_{REF}$
I_{TL}	1 to 0 Transition Current (QBD Pins)		−650	µA	V_{IN} = 2.0V
I_{IL}	Logical 0 Input Current (QBD Pins)		−50	µA	V_{IN} = 0.45V
I_{IL1}	Logical 0 Input Current in Reset (Note 2) (ALE, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, INST, P2.0, Port 3/4)		−850	µA	V_{IN} = 0.45 V

NOTE:
1. All pins except RESET and XTAL1.
2. Holding these pins below V_{IH} in Reset may cause the part to enter test modes.

D.C. Characteristics (Over specified operating conditions) (Continued)

Symbol	Description	Min	Typ(7)	Max	Units	Test Conditions
I_{CC}	Active Mode Current in Reset		40	55	mA	XTAL1 = 12 MHz
I_{REF}	A/D Converter Reference Current		2	5	mA	$V_{CC} = V_{PP} = V_{REF} = 5.5V$
I_{IDLE}	Idle Mode Current		10	22	mA	
I_{CC1}	Active Mode Current (Typical)		15	22	mA	XTAL1 = 3.5 MHz
I_{PD}(8)	Powerdown Mode Current		5		μA	$V_{CC} = V_{PP} = V_{REF} = 5.5V$
R_{RST}	Reset Pullup Resistor	6K		50K	Ω	
C_S	Pin Capacitance (Any Pin to V_{SS})			10	pF	f_{TEST} = 1.0 MHz

NOTES:
(Notes apply to all specifications)
1. QBD (Quasi-bidirectional) pins include Port 1, P2.6 and P2.7.
2. Standard Outputs include AD0–15, $\overline{RD}$, $\overline{WR}$, ALE, $\overline{BHE}$, INST, HSO pins, PWM/P2.5, CLKOUT, RESET, Ports 3 and 4, TXD/P2.0, and RXD (in serial mode 0). The V_{OH} specification is not valid for RESET. Ports 3 and 4 are open-drain outputs.
3. Standard Inputs include HSI pins, CDE, EA, READY, BUSWIDTH, NMI, RXD/P2.1, EXTINT/P2.2, T2CLK/P2.3, and T2RST/P2.4.
4. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V or V_{OH} is held below V_{CC} − 0.7V:
   I_{OL} on Output pins: 10 mA
   I_{OH} on quasi-bidirectional pins: self limiting
   I_{OH} on Standard Output pins: 10 mA
5. Maximum current per bus pin (data and control) during normal operation is ±3.2 mA.
6. During normal (non-transient) conditions the following total current limits apply:

Port 1, P2.6	I_{OL}: 29 mA	I_{OH} is self limiting
HSO, P2.0, RXD, $\overline{RESET}$	I_{OL}: 29 mA	I_{OH}: 26 mA
P2.5, P2.7, $\overline{WR}$, $\overline{BHE}$	I_{OL}: 13 mA	I_{OH}: 11 mA
AD0–AD15	I_{OL}: 52 mA	I_{OH}: 52 mA
$\overline{RD}$, ALE, INST–CLKOUT	I_{OL}: 13 mA	I_{OH}: 13 mA

7. Typicals are based on a limited number of samples and are not guaranteed. The values listed are at room temperature and $V_{REF} = V_{CC} = 5V$.
8. I_{PD} is not guaranteed on the standard 87C196KB part and may exceed 100 μA on some parts. Customers whose applications use the power down mode and require a guaranteed maximum value of I_{PD} should order the S-spec part, contact an Intel Field Representative.

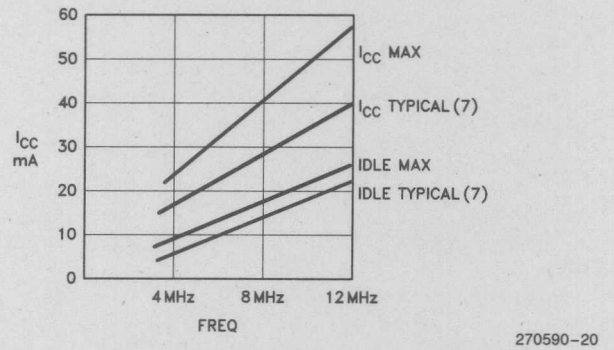


Figure 15. I_{CC} and I_{IDLE} vs. Frequency

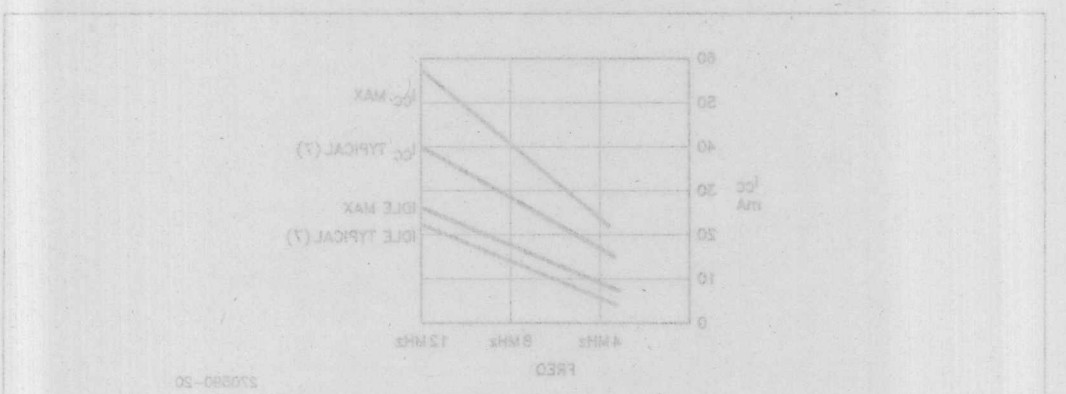
A.C. Characteristics (Over specified operating conditions)

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The system must meet these specifications to work with the 87C196KB:

Symbol	Description	Min	Max	Units	Notes
T_{AVYV}	Address Valid to READY Setup		$2T_{OSC} - 85$	ns	
T_{LLYV}	ALE Low to READY Setup: 87C196KB10 87C196KB12		 $T_{OSC} - 80$ $T_{OSC} - 72$	 ns ns	
T_{YLYH}	NonREADY Time	No upper limit		ns	
T_{CLYX}	READY Hold after CLKOUT Low	0	$T_{OSC} - 30$	ns	(Note 1)
T_{LLYX}	READY Hold after ALE Low	$T_{OSC} - 15$	$2T_{OSC} - 40$	ns	(Note 1)
T_{AVGV}	Address Valid to Buswidth Setup		$2T_{OSC} - 85$	ns	
T_{LLGV}	ALE Low to Buswidth Setup		$T_{OSC} - 70$	ns	
T_{CLGX}	Buswidth Hold after CLKOUT Low	0		ns	
T_{AVDV}	Address Valid to Input Data Valid 80C196 KB10 80C196 KB12		 $3T_{OSC} - 70$ $3T_{OSC} - 67$	 ns ns	
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid 87C196KB10 87C196KB12		 $T_{OSC} - 30$ $T_{OSC} - 23$	 ns ns	
T_{CLDV}	CLKOUT Low to Input Data Valid		$T_{OSC} - 50$	ns	
T_{RHDZ}	End of $\overline{RD}$ to Input Data Float		$T_{OSC} - 20$	ns	
T_{RXDX}	Data Hold after $\overline{RD}$ Inactive	0		ns	

NOTES:
1. If max is exceeded, additional wait states will occur.



A.C. Characteristics (Over specified operating conditions) (Continued)

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The 80C196KA will meet these specifications:

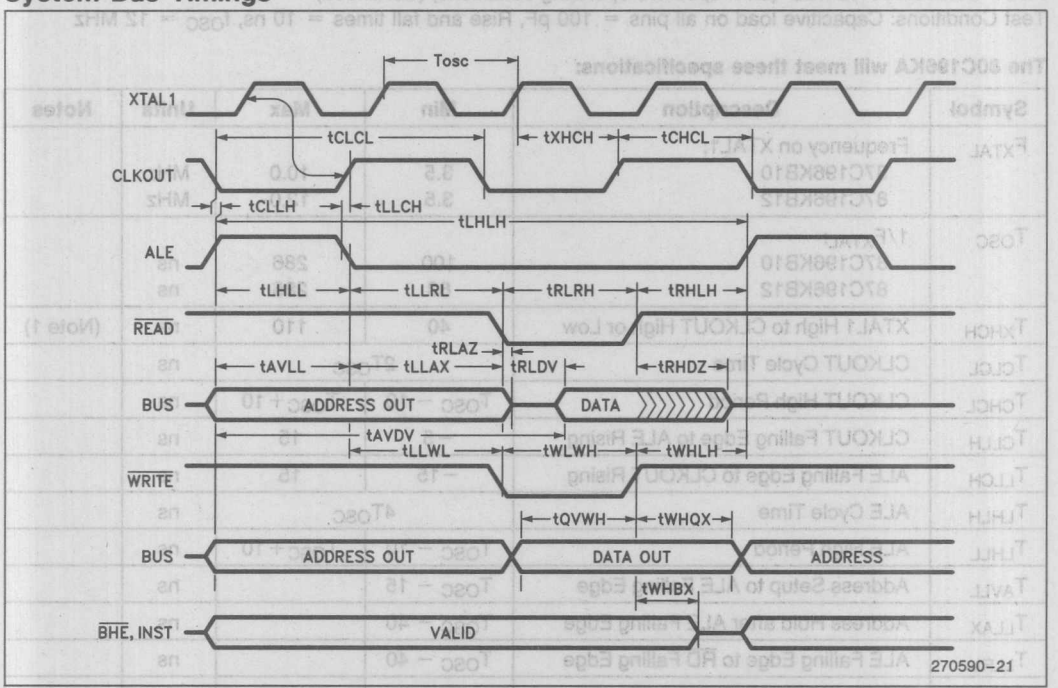
Symbol	Description	Min	Max	Units	Notes
F_{XTAL}	Frequency on XTAL1:				
	87C196KB10	3.5	10.0	MHz	
	87C196KB12	3.5	12.0	MHz	
T_{OSC}	$1/F_{XTAL}$:				
	87C196KB10	100	286	ns	
	87C196KB12	83	286	ns	
T_{XHCH}	XTAL1 High to CLKOUT High or Low	40	110	ns	(Note 1)
T_{CLCL}	CLKOUT Cycle Time	$2T_{OSC}$		ns	
T_{CHCL}	CLKOUT High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{CLLH}	CLKOUT Falling Edge to ALE Rising	−5	15	ns	
T_{LLCH}	ALE Falling Edge to CLKOUT Rising	−15	15	ns	
T_{LHLH}	ALE Cycle Time	$4T_{OSC}$		ns	
T_{LHLL}	ALE High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{AVLL}	Address Setup to ALE Falling Edge	$T_{OSC} - 15$		ns	
T_{LLAX}	Address Hold after ALE Falling Edge	$T_{OSC} - 40$		ns	
T_{LLRL}	ALE Falling Edge to $\overline{RD}$ Falling Edge	$T_{OSC} - 40$		ns	
T_{RLCL}	$\overline{RD}$ Low to CLKOUT Falling Edge	10	30	ns	
T_{RLRH}	$\overline{RD}$ Low Period	$T_{OSC} - 5$		ns	
T_{RHLH}	$\overline{RD}$ Rising Edge to ALE Rising Edge	T_{OSC}	$T_{OSC} + 25$	ns	(Note 2)
T_{RLAZ}	$\overline{RD}$ Low to Address Float		10	ns	
T_{LLWL}	ALE Falling Edge to $\overline{WR}$ Falling Edge	$T_{OSC} - 10$		ns	
T_{CLWL}	CLKOUT Low to $\overline{WR}$ Falling Edge	0	25	ns	
T_{QVWH}	Data Stable to $\overline{WR}$ Rising Edge:				
	87C196KB10	$T_{OSC} - 30$		ns	
	87C196KB12	$T_{OSC} - 23$		ns	
T_{CHWH}	CLKOUT High to $\overline{WR}$ Rising Edge	−10	10	ns	
T_{WLWH}	$\overline{WR}$ Low Period	$T_{OSC} - 30$		ns	
T_{WHQX}	Data Hold after $\overline{WR}$ Rising Edge	$T_{OSC} - 10$		ns	
T_{WHLH}	$\overline{WR}$ Rising Edge to ALE Rising Edge	$T_{OSC} - 10$	$T_{OSC} + 15$	ns	(Note 2)
T_{WHBX}	$\overline{BHE}$, INST HOLD after $\overline{WR}$ Rising Edge	$T_{OSC} - 10$		ns	

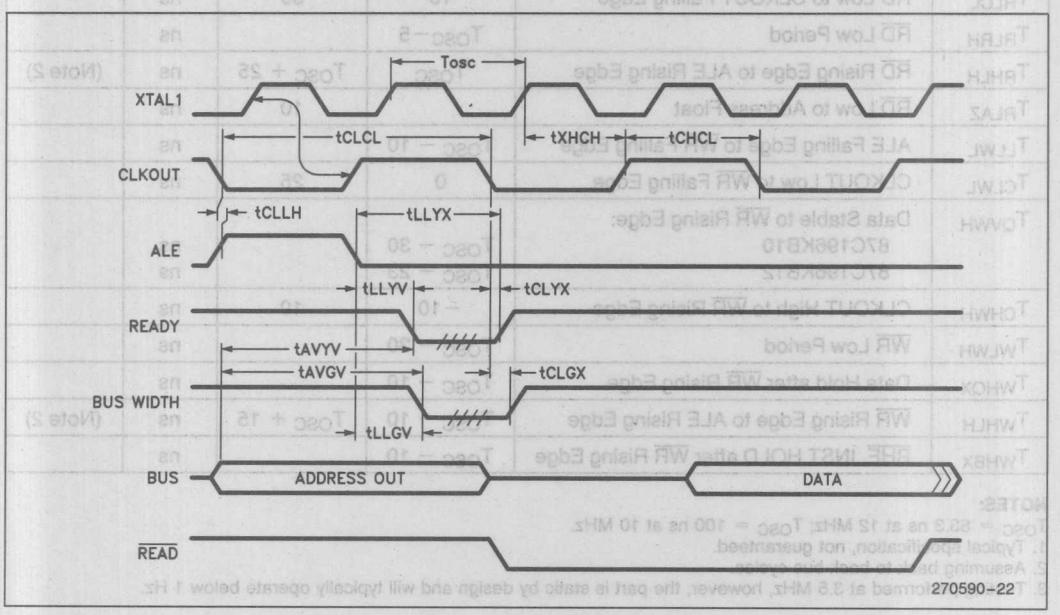
NOTES:

T_{OSC} = 83.3 ns at 12 MHz; T_{OSC} = 100 ns at 10 MHz.

1. Typical specification, not guaranteed.
2. Assuming back-to-back bus cycles.
3. Testing performed at 3.5 MHz, however, the part is static by design and will typically operate below 1 Hz.

System Bus Timings

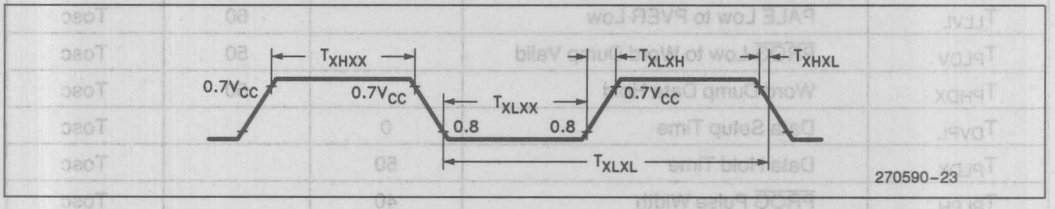




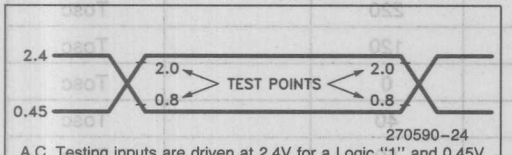
EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
$1/T_{XLXL}$	Oscillator Frequency	3.5	12	MHz
T_{XLXL}	Oscillator Period (T_{OSC})	83	286	ns
T_{XHXX}	High Time	32		ns
T_{XLXX}	Low Time	32		ns
T_{XLXH}	Rise Time		10	ns
T_{XHXL}	Fall Time		10	ns

EXTERNAL CLOCK DRIVE WAVEFORMS

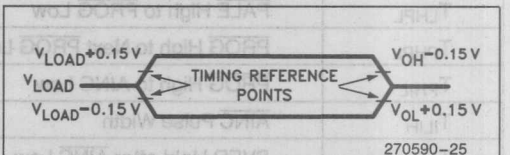


A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} = \pm 15$ mA.

EXPLANATION OF AC SYMBOLS

Each symbol is two pairs of letters prefixed by "T" for time. The characters in a pair indicate a signal and its condition, respectively. Symbols represent the time between the two signal/condition points.

Conditions:

H - High
L - Low
V - Valid
X - No Longer Valid
Z - Floating

Signals:

A - Address
B - $\overline{BHE}$
C - CLKOUT
D - DATA
G - Buswidth
L - ALE/$\overline{ADV}$
R - $\overline{RD}$
W - $\overline{WR}/\overline{WRH}/\overline{WRL}$
X - XTAL1
Y - READY

A.C. EPROM Programming Characteristics(1)

Operating Conditions: Load Capacitance = 150 pF, T_A = +25°C ±5°C, V_{CC}, V_{REF} = 5V, V_{SS}, ANGND = 0V, V_{PP} = 12.75V ± 0.25V, EA = 12.75V ± 0.25

Symbol	Description	Min	Max	Units
T_{SHLL}	Reset High to First $\overline{PALE}$ Low	1100		Tosc
T_{LLLH}	$\overline{PALE}$ Pulse Width	40		Tosc
T_{AVLL}	Address Setup Time	0		Tosc
T_{LLAX}	Address Hold Time	50		Tosc
T_{LLVL}	$\overline{PALE}$ Low to PVER Low		60	Tosc
T_{PLDV}	$\overline{PROG}$ Low to Word Dump Valid		50	Tosc
T_{PHDX}	Word Dump Data Hold		50	Tosc
T_{DVPL}	Data Setup Time	0		Tosc
T_{PLDX}	Data Hold Time	50		Tosc
T_{PLPH}	$\overline{PROG}$ Pulse Width	40		Tosc
T_{PHLL}	$\overline{PROG}$ High to Next $\overline{PALE}$ Low	120		Tosc
T_{LHPL}	$\overline{PALE}$ High to $\overline{PROG}$ Low	220		Tosc
T_{PHPL}	$\overline{PROG}$ High to Next $\overline{PROG}$ Low	120		Tosc
T_{PHIL}	$\overline{PROG}$ High to AINC Low	0		Tosc
T_{ILIH}	$\overline{AINC}$ Pulse Width	40		Tosc
T_{ILVH}	PVER Hold after $\overline{AINC}$ Low	50		Tosc
T_{ILPL}	$\overline{AINC}$ Low to $\overline{PROG}$ Low	170		Tosc
T_{PHVL}	$\overline{PROG}$ High to PVER Low		90	Tosc

NOTES:
1. Timings are based on theoretical calculations.
2. Run Time Programming is done with Fosc = 6.0 MHz to 12.0 MHz, V_{REF} = 5V ± 0.65V. T_A = +25°C to ±5°C and V_{PP} = 12.75V. For run-time programming over a full operating range, contact the factory.

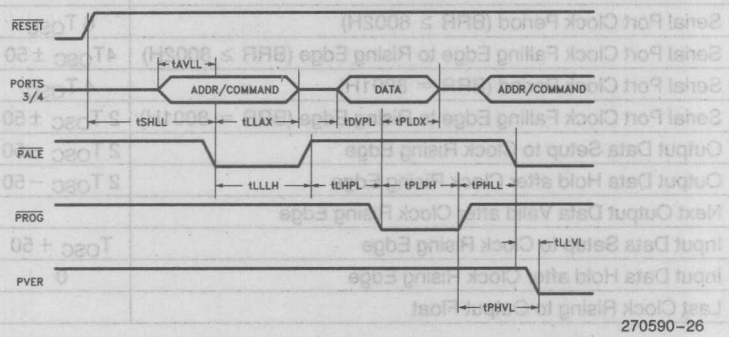
D.C. EPROM Programming Characteristics

Symbol	Description	Min	Max	Units
I_{PP}	V_{PP} Supply Current (When Programming)		100	mA

NOTE:
V_{PP} must be within 1V of V_{CC} while V_{CC} < 4.5V. V_{PP} must not have a low impedance path to ground or V_{SS} while V_{CC} > 4.5V.

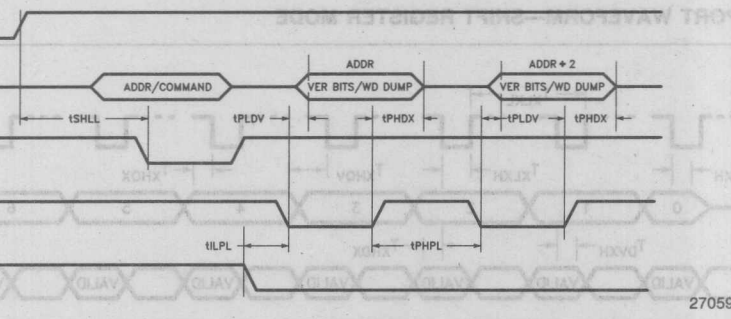
EPROM PROGRAMMING WAVEFORMS

SLAVE PROGRAMMING MODE DATA PROGRAM MODE WITH SINGLE PROGRAM PULSE



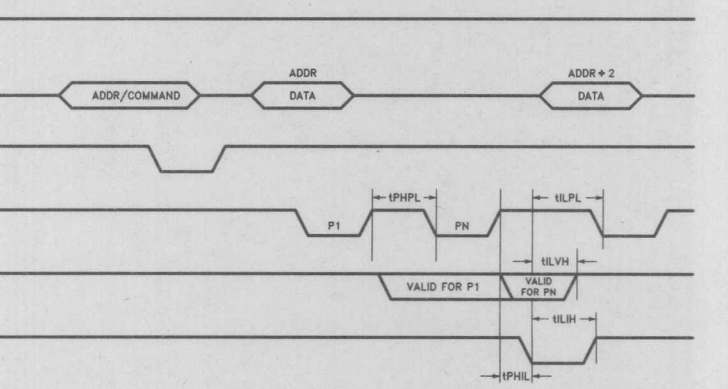
270590-26

SLAVE PROGRAM MODE IN WORD DUMP OR DATA VERIFY MODE WITH AUTO INCREMENT



270590-27

SLAVE PROGRAMMING MODE TIMING IN DATA PROGRAM MODE WITH REPEATED PROG PULSE AND AUTO INCREMENT



270590-28

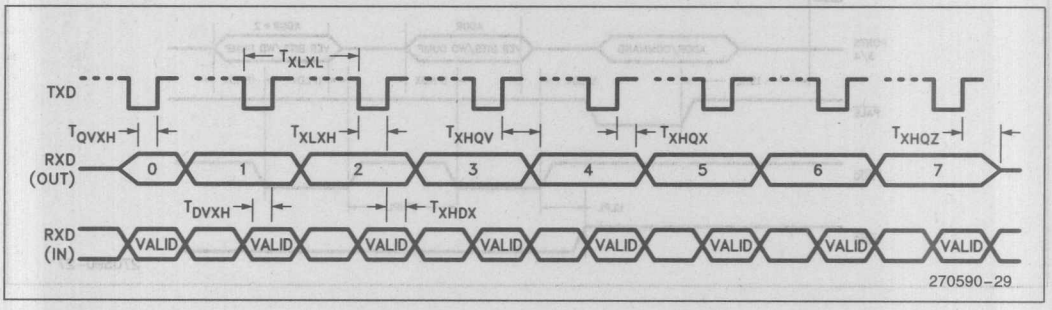
A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT TIMING—SHIFT REGISTER MODE

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period (BRR ≥ 8002H)	6 T_{OSC}		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge (BRR ≥ 8002H)	4T_{OSC} ±50		ns
T_{XLXL}	Serial Port Clock Period (BRR = 8001H)	4 T_{OSC}		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge (BRR = 8001H)	2 T_{OSC} ±50		ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	2 T_{OSC} −50		ns
T_{XHQX}	Output Data Hold after Clock Rising Edge	2 T_{OSC} −50		ns
T_{XHQV}	Next Output Data Valid after Clock Rising Edge		2 T_{OSC} +50	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	T_{OSC} +50		ns
T_{XHDX}	Input Data Hold after Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		1 T_{OSC}	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



NOTE:
1. Timings are not tested but guaranteed by design.

A TO D CHARACTERISTICS

There are two modes of A/D operation: with or without clock prescaler. The modes are shown in the table below. The maximum frequency with the clock prescaler off is 8 MHz.

The converter is ratiometric, so the absolute accuracy is directly dependent on the accuracy and stability of V_{REF}. V_{REF} must be close to V_{CC} since it supplies both the resistor ladder and the digital section of the converter.

A/D CONVERTER SPECIFICATIONS

The specifications given below assume adherence to the Operating Conditions section of this data sheet. Testing is performed with the clock prescaler on and with V_{REF} = 5.12V and 10 MHz on XTAL1.

Clock Prescaler On IOC2.4 = 0	Clock Prescaler Off IOC2.4 = 1
158 States 26.33 µs @ 12 MHz	91 States 22.75 µs @ 8 MHz, 18.2 µs @ 10 MHz

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		512(4)	1024 10	Levels Bits	
Absolute Error		0	±8	LSBs	
Full Scale Error	0/−2			LSBs	
Zero Offset Error	−0.5/2.0			LSBs	5
Non-Linearity		0	±8	LSBs	1
Differential Non-Linearity		0	±2	LSBs	1
Channel-to-Channel Matching		0	±1	LSBs	1
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.009 0.009 0.009			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation		−60		dB	1, 2, 3
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage			±3.0	µA	
Sample Time: Prescaler On Prescaler Off	 15 8			 State State	 4 4
Sample Capacitive	3			pF	

NOTES:
* These values are expected for most parts at 25°C but are not tested or guaranteed.
**An "LSB", as used here, has a value of approximately 5 mV.
1. These values are not tested in production and are guaranteed based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. Multiplexer Break-Before-Make Guaranteed.
4. One state = 167 ns at 12 MHz, 250 ns at 8 MHz.
5. A large positive offset error can cause codes above 1020 to be missed. This error is also reflected in the Absolute, Full Scale, Zero Offset and Non-Linearity specifications.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK BEFORE MAKE—The property of a multiplexer which guarantees that a previously selected channel is selected (i.e., the converter will not short inputs together.

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For an 8-bit converter with a reference voltage of 5.12V, one LSB is 20 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 40 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 80 mV.)

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE TIME—The time that the sample window is open.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

87C196KB FUNCTIONAL DEVIATIONS

The 87C196KB has the following problems.

1. Interrupts will not occur between an untaken conditional jump and the next instruction. A series of untaken jumps will hold off interrupts until after the last untaken jump.
2. The DJNZW instruction is not guaranteed to be functional. The instruction, if encountered, will not cause an unimplemented opcode. The DJNZ (byte instruction) works correctly and should be used instead.
3. The serial port only tolerates a +1.25%, −7.5% baud rate error between transmitter and receiver. If the serial port fails on the receiver, increase the baud rate.
4. The CDE function is not guaranteed to work. In order to ensure proper 87C196KB operation, the pin must be grounded.

DIFFERENCES BETWEEN THE 80C196KA AND THE 80C196KB

The 8XC196KB is identical to 8XC196KA except for the following differences.

1. ALE is high after reset on the 87C196KB instead of low like the 80C196KA.
2. The DJNZW instruction is not guaranteed to work on the 87C196KB.
3. The $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ bus protocol is available on the 87C196KB.

CONVERTING FROM OTHER 8096 FAMILY PRODUCTS TO THE 8XC196KB

The following list of suggestions for designing an 879XBH system will yield a design that is easily converted to the 87C196KB.

1. Do not base critical timing loops on instruction or peripheral execution times.
2. Use equate statements to set all timing parameters, including the baud rate.
3. Do not base hardware timings on CLKOUT or XTAL1. The timings of the 80C196KB are different than those of the 8X9XBH, but they will function with standard ROM/EPROM/Peripheral type memory systems.
4. Verify that all inputs are driven high or low and not left floating.
5. Indexed and indirect operations relative to the stack pointer (SP) work differently on the 80C196KB than on the 8096. On the 8096, the address is calculated based on the un-updated version of the stack pointer. The 80C196KB uses the updated version. The offset for PUSH[SP], POP[SP], PUSH nn[SP] and POP nn[SP] instructions may need to be changed by a count of 2.
6. The 87C196KB does not support gang programming in auto programming mode. Gang programming in slave programming is supported.
7. $\overline{\text{PACT}}$ has changed from the HSO.O on the 8796BH to P2.7 on the 87C196KB.

REVISION HISTORY

The following differences exist between the 87C196KB advanced and 87C196KB preliminary data sheets.

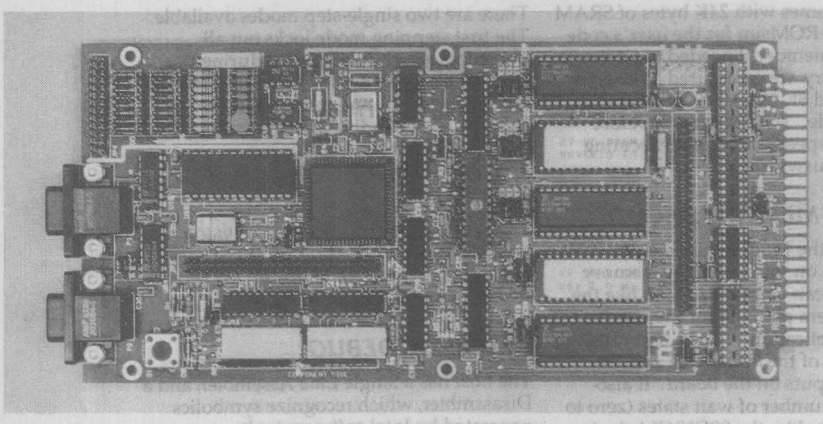
1. The A/D no sample and hold feature is no longer available.
2. With the clock prescaler disabled, the A/D has a large absolute error for frequencies greater than 8 MHz. The clock prescaler should be enabled for frequencies greater than 8 MHz.
3. The following A/D characteristics have changed

	ADVANCED	PRELIMINARY
Absolute Error	−4/+4	±8
Full Scale Error	−0.5/±0.5	0/−2
Zero Offset	±0.5	−0.5/+2
Non Linearity	−4/+4	±8

4. The I_{IL1} Logical 0 input current in reset changed from −500 μA to −850 μA.
5. The T_{LLYV} (ALE Low to Ready Setup) has changed to T_{OSC} −72 at 12 MHz and T_{OSC} − 80 at 10 MHz.
6. T_{WHBX} has changed to T_{OSC} − 10.
7. New current specifications have been added to reflect typical current consumption at room temperature.

8. Current curves for I_{IDLE}(Max) and I_{IDLE}(Typical) have been added.
9. A new specification T_{RLAZ} (RD low to Address Float) has been specified for 10 ns.
10. The minimum instruction times for some of the indirect instructions which write to external memory have changed.
11. Serial port timings have been added.
12. A/D glossary terms have been removed.
13. A/D sample time and sample capacitance specifications have been added.
14. The upper four bits of the A/D command should be set to zero for future compatibility.
15. HSO commands 0C and 0D should not be used to ensure future compatibility.
16. The Modified Quick Pulse™ algorithm has been recommended to ensure programming of the EPROM over the life of the part for both run-time and slave programming modes.
17. T_{AVYV} (Address valid to READY setup) has been changed to $2T_{OSC} - 0.85$.
18. T_{AVGV} (Address valid to BUSWIDTH setup) has been changed to $2T_{OSC} - 0.85$.
19. T_{QVWH} (Data Stable to $\overline{WR}$ Rising Edge) has changed to $T_{OSC} - 23$ at 12 MHz and $T_{OSC} - 30$ at 10 MHz.
20. T_{AVDV} (Address Valid to Input Data Valid) has been changed to $3T_{OSC} - 70$ at 10 MHz and $3T_{OSC} - 67$ at 12 MHz.

EV80C196KA EVALUATION BOARD



LOW COST CODE EVALUATION TOOL

Intel's EV80C196KA evaluation board provides a hardware environment for code execution and software debugging at a relatively low cost. The board features the 80C196KA advanced, CHMOS*, 16-bit microcontroller, the newest member of the industry standard 8096 family. The board allows the user to take full advantage of the power of the 8096. The EV80C196KA provides zero wait-state, 12 MHz execution of a user's code. Plus, its memory (ROMsim) can be reconfigured to match the user's planned memory system, allowing for exact analysis of code execution speeds in a particular application.

Popular features such as a symbolic single line assembler/disassembler, single-step program execution, and sixteen software breakpoints are standard on the EV80C196KA. Intel provides a complete code development environment using assembler (ASM-96) as well as high-level languages such as Intel's iC-96 or PL/M-96 to accelerate development schedules.

The evaluation board is hosted on an IBM PC** or BIOS-compatible clone, already a standard development solution in most of today's engineering environments. The source code for the on-board monitor (written in ASM-96) is public domain. The program is about 1K, and can be easily modified to be included in the user's target hardware. In this way, the provided PC host can be used throughout the development phase.

EV80C196KA FEATURES

- Zero Wait-State 12 MHz Execution Speed
- 24K Bytes of ROMsim
- Flexible Wait-State, Buswidth, Chip-Select controller
- Totally CMOS, low power board
- Concurrent Interrogation of Memory and Registers
- Sixteen Software Breakpoints
- Two Single Step Modes
- High-Level Language Support
- Symbolic Debug
- RS-232-C Communication Link

FULL SPEED EXECUTION

The EV80C196KA executes the user's code from on-board ROMsim at 12 MHz with zero wait-states. By changing crystals on the 80C196KA any slower execution speed can be evaluated. The boards host interface timing is not affected by this crystal change.

intel

*CHMOS is a patented Intel process.
**IBM PC, XT, AT and DOS are registered trademarks of International Business Machines Corporation.
Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

MARCH 1988
Order number: 270591-001

© Intel Corporation 1988

24K BYTES OF ROMSIM

The board comes with 24K bytes of SRAM to be used as ROMsim for the user's code and as data memory if needed. 16K bytes of this memory are configured as sixteen bits wide, and 8K bytes are configured as eight bits wide. The user can therefore evaluate the speed of the part executing from either buswidth.

FLEXIBLE MEMORY DECODING

By changing the Programable Logic Device (PLD) on the board, the memory on the board can be made to look like the memory system planned for the user's hardware application. The PLD controls the buswidth of the 80C196KA and the chip-select inputs on the board. It also controls the number of wait states (zero to three) generated by the 80C196KA during a memory cycle. These features can all be selected with 64 byte boundaries of resolution.

TOTALLY CMOS BOARD

The EV80C196KA board is built totally with CMOS components. Its power consumption is therefore very low, requiring 5 volts at only 425 mA. If the on board LEDs are disabled, the current drops to only 150 mA. The board also requires +/- 12 volts at 10 mA.

CONCURRENT INTERROGATION OF MEMORY AND REGISTERS

The monitor for the EV80C196KA allows the user to read and modify internal registers and external memory while the user's code is running in the board.

SIXTEEN SOFTWARE BREAKPOINTS

There are sixteen breakpoints available which automatically substitute a TRAP instruction for a user's instruction at the breakpoint location. The substitution occurs when execution is started. If the code is halted or a breakpoint is reached, the user's code is restored in the ROMsim.

TWO STEP MODES

There are two single-step modes available. The first stepping mode locks out all interrupts which might occur during the step. The second mode enables interrupts, and treats subroutine calls and interrupt routines as one indivisible instruction.

HIGH LEVEL LANGUAGE SUPPORT

The host software for the EV80C196KA board is able to load absolute object code generated by ASM-96, iC-96, PL/M-96 or RL-96 all of which are available from Intel.

SYMBOLIC DEBUG

The host has a Single Line Assembler, and a Disassmbler, which recognize symbolics generated by Intel software tools.

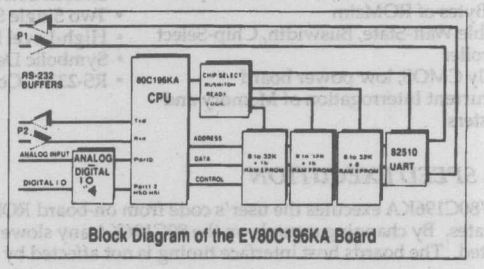
RS-232-C COMMUNICATION LINK

The EV80C196KA communicates with the host using an Intel 82510 UART provided on board. This frees the on-chip UART of the 80C196KA for the user's application.

PERSONAL COMPUTER REQUIREMENTS

The EV80C196KA Evaluation Board is hosted on an IBM PC**, XT**, AT** or BIOS-compatible clone. The PC must meet the following minimum requirements:

- 512K Bytes of Memory
- One 360K Byte floppy Disk Drive
- PC DOS** 3.1 or Later
- A Serial Port (COM1 or COM2) at 9600 Baud
- ASM-96, iC-96 or PL/M-96
- A text editor such as AEDIT



Block Diagram of the EV80C196KA Board

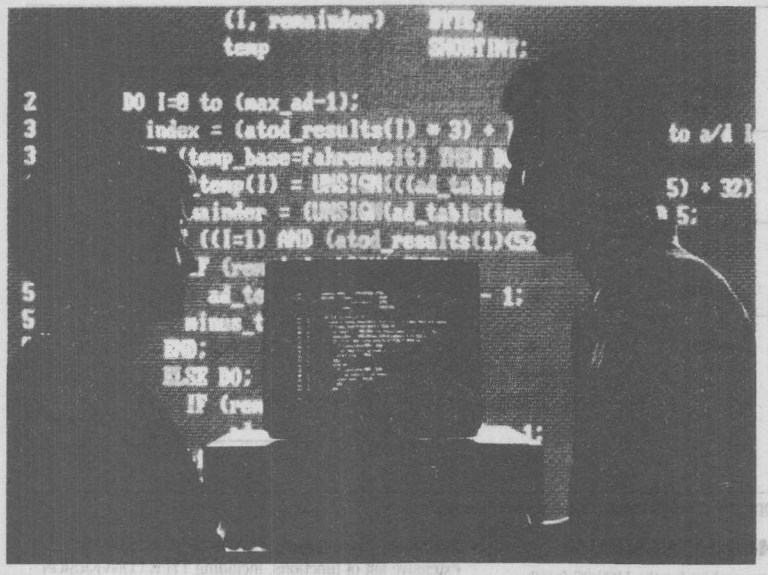
Printed in USA/TP270/0388/10K/SM
Embedded Controllers

MCS®-96 Development Support Tools

7

7 MCS®-96 Development Support Tools

8096/196 Software Development Packages



COMPLETE SOFTWARE DEVELOPMENT SUPPORT FOR THE 8096/196 FAMILY OF MICROCONTROLLERS

Intel supports application development for its 8096 and 80C196 family of microcontrollers with a complete set of development languages and utilities. These tools include a macroassembler, a PL/M compiler, a C compiler, linker/relocator program, floating point arithmetic library, a librarian utility, and an object-to-hex utility. Develop code in the language(s) you desire, then combine object modules from different languages into a single, fast program.

FEATURES

- Software Tools support all members of Intel's MCS®-96 family
- ASM-96/196 macroassembler for speed critical code
- PL/M-96/196 package for the maintainability and reliability of a high-level language with support for many low-level hardware functions
- iC-96/196 package for structured C language programming, with many hardware specific extensions
- Linker/Relocator program for linking modules generated in assembler, PL/M or C and assigning absolute addresses to relocatable code. RL-96 prepares your code for execution in target with a simple, one-step operation
- 32-bit Floating Point Arithmetic Library to reduce your development effort and to allow fast, highly optimized numerics-intensive processing
- Library utility for creating and maintaining software object module libraries
- PROM building utility that converts object modules into standard hexadecimal format for easy download into a non-Intel PROM Programmer
- Hosted on IBM PC XT/AT with PC-DOS 3.0 or above

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1988

September, 1988
Order Number: 280793-002

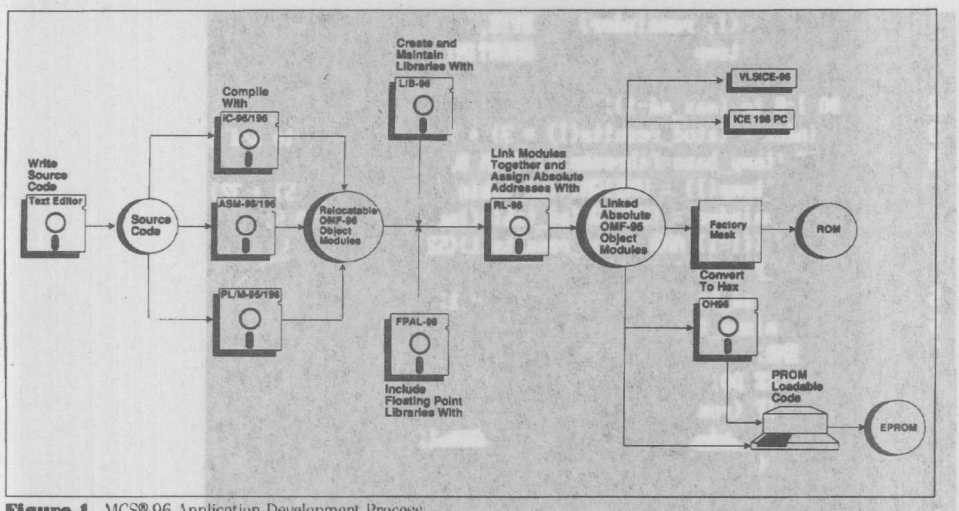


Figure 1. MCS®-96 Application Development Process

ASM-96/196 MACROASSEMBLER

ASM-96/196 is the macroassembler for the MCS-96 family of microcontrollers, including the 80C196. ASM-96/196 translates symbolic assembly language mnemonics into relocatable object code.

The macro facility in ASM-96/196 saves development and maintenance time, since common code sequences need only be developed once. The assembler also supports symbolic access to the many features of the 8096/196 and provides an "include" file with all 8096/196 registers defined.

PL/M-96/196 SOFTWARE PACKAGE

PL/M-96/196 is a high-level programming language designed to support the software requirements of advanced 16-bit microcontrollers. The PL/M-96/196 compiler translates PL/M high-level language statements into 8096/196 relocatable object code. Major features of the PL/M-96/196 compiler include:

- **Structured programming.** The PL/M language supports modular and structured programming, making programs easier to understand, maintain, and debug.
- **Built-in functions.** PL/M-96/196 includes an extensive list of functions, including TYPE CONVERSION functions, STRING manipulations, and functions for interrogating MCS-96 hardware flags.
- **Interrupt handling.** The INTERRUPT attribute allows you to define interrupt handling procedures. The compiler generates code to save and restore the program status word for INTERRUPT procedures.
- **Compiler controls.** Compile-time options increase the flexibility of the PL/M-96/196 compiler. These controls include: optimization, conditional compilation, the inclusion of common PL/M source files from disk, cross-reference of symbols, and optional assembly language code in the listing file.
- **Data types.** PL/M-96/196 supports seven data types, allowing PL/M-96/196 to perform three different kinds of arithmetic: signed, unsigned, and floating point.
- **Language compatibility.** PL/M-96/196 object modules are compatible with all other object modules generated by Intel MCS-96 translators.

iC-96/196 SOFTWARE PACKAGE

Intel's iC-96/196 is a structured programming language designed to support applications for the 16-bit family of MCS-96 microcontrollers. iC-96/196 implements the C language as described in the Kernighan and Ritchie book, *The C Programming Language*, and includes many of the enhancements as defined by the proposed ANSI C standard. Major features of the iC-96/196 compiler include:

- **Symbolics.** The iC-96/196 compiler boosts programmer productivity by providing extensive debug information, including symbols. The debug information can be used to debug the code using either the VLSiCE™-96 emulator or the ICE™-196PC emulator.
- **Architecture Support.** iC-96/196 generates code which is fully optimized for the MCS-96 architecture. iC-96/196 provides an INTERRUPT attribute, allowing you to define interrupt handling functions in C, and library routines which allow you to enable and disable interrupts directly from C (mid-1989). A REENTRANT/ NOREENTRANT control is also included, allowing the compiler to identify non-reentrant procedures. This gives you full access to the large MCS-96 register set.
- **Standard language.** iC-96/196 accepts standard C source code. iC-96/196 code is fully linkable with both PL/M-96/196 and ASM-96/196 modules via an "alien" attribute, allowing programmers to utilize the optimal language for any application. In addition, programmers can quickly begin programming with iC-96/196 because it conforms to accepted C language standards.

RL-96/196 LINKER/RELOCATOR

Intel's RL-196 utility is used to link multiple MCS-96 object modules into a single program and then assign absolute addresses to all relocatable addresses in the new program. Modules can be written in ASM-96/196, PL/M-96/196, or iC-96/196.

The RL-96/196 utility also promotes programmer productivity by encouraging modular programming. Because applications can be broken into separate modules, they're easier to design, test and maintain. Standard modules can be reused in different applications, saving software development time.

FPAL-96/196 FLOATING POINT ARITHMETIC LIBRARY

FPAL-96/196 is a library of single-precision 32-bit floating point arithmetic functions. These functions are compatible with the IEEE floating point standard for accuracy and reliability and include an error-handler library.

LIB-96/196

The Intel LIB-96/196 utility creates and maintains libraries of software object modules. Standard modules can be placed in a library, and linked into your applications programs using RL-96/196.

OH-96/196

The OH-96/196 utility converts Intel OMF-96 object modules into standard hexadecimal format. This allows the code to be loaded directly into a PROM via non-Intel PROM programmers.

SERVICE, SUPPORT, AND TRAINING

Intel augments its MCS-96 architecture family development tools with a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hot-line support; and software and hardware maintenance contracts. This full line of services will ensure your design success.

ORDERING INFORMATION

D86ASM96NL*	96/196 Assembler for PC XT or AT system (or compatible), running DOS 3.0 or higher	D86C96NL*	iC-96/196 Software Package for PC XT or AT system (or compatible), running DOS 3.0 or higher
D86PLM96NL*	PL/M-96/196 Software Package for PC XT or AT system (or compatible), running DOS 3.0 or higher		

*Also Includes: Relocator/Linker, Object-to-hex converter, Floating Point Arithmetic Library, and Librarian.

VLSICE™-96 IN-CIRCUIT EMULATOR

IN-CIRCUIT EMULATOR FOR THE 8X9X FAMILY OF MICROCONTROLLERS

The VLSiCE™-96 emulator is a complete hardware/software debug environment for developing systems based on the Intel 8x9x family of microcontrollers. The VLSiCE-96 emulator supports all NMOS members of Intel's MCS-96 microcontrollers, including the 8096BH, the 8098, the 8095, the 8097, and the 8096-90. With high performance 12 MHz emulation, symbolic debugging, and flexible memory mapping, the VLSiCE-96 emulator expedites all stages of development: software development, hardware development, system integration and system test.

FEATURES

- Real-time transparent emulation, up to 12 MHz
- 64K of mappable memory to allow early software debug and (EP)ROM simulation, even before any target hardware is available.
- Trace contains execution address, opcode, symbolics, and bus information
- 4K frame trace buffer for storing real-time execution history
- Ability to break or trace on execution addresses, opcodes, data values, or flags values
- Symbolic debugging for faster and easier access to memory location and program variables.
- Fast breaks and dynamic trace to allow the user to modify and interrogate memory, and access the trace buffer without stopping emulation.
- On-line Help file to speed development
- Shadow Registers can read many write-only registers and write to many read-only registers, allowing enhanced debugging over component features
- Includes 68-pin PGA adaptor; optional 68-pin PLCC and 48-pin DIP adaptors are also available
- Serially hosted on IBM PC AT/XT or compatibles with DOS 3.0 or greater.

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

September, 1988
Order Number: 280794-002

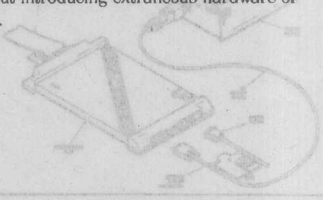
© Intel Corporation 1988

ONE TOOL FOR ENTIRE DEVELOPMENT CYCLE

The VLSiCE-96 emulator speeds target system development by allowing hardware and software design to proceed simultaneously. You can develop software even before prototype hardware is finished. And because the VLSiCE-96 emulator precisely matches the component's electrical and timing characteristics, it's a valuable tool for hardware development and debug.

The VLSiCE-96 emulator also simplifies and expedites system integration and test. As each section of the hardware is completed, it is simply added to the prototype and tested in real-time. When the prototype is complete, it is tested with the final version of the system software. The VLSiCE-96 emulator can then be used to verify or debug the target system as a completed unit.

Because it supports the ROMless, ROM and EPROM versions of Intel's microcontrollers, the VLSiCE-96 emulator can debug a prototype or production product at any stage in its development without introducing extraneous hardware or software test tools.



SPECIFICATIONS

HOST REQUIREMENTS

An IBM PC AT/XT or compatible with 512K bytes RAM and hard disk. Intel recommends an IBM PC AT or compatible with 640K bytes of RAM, one floppy drive and one hard disk running PC-DOS 3.1 or later.

System Performance

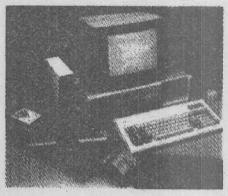
Mappable zero wait state (up to 12 MHz) Min 0K bytes, Max 64K bytes	Mappable to user memory or ICE memory in 1K blocks on 1K boundaries
Trace Buffer	4K bytes × 48 bits
Virtual Symbol Table	A maximum of 61K bytes of host memory space is available for the virtual symbol table (VST). The rest of the VST resides on disk and is paged in and out of host memory as needed.

Electrical Characteristics

Power Supply
100V-120V or 200V-240V (selectable)
50 Hz-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

Physical Characteristics

Controller Pod	
Width:	8¼" (21 cm)
Height:	1½" (4 cm)
Depth:	13½" (34 cm)
Weight:	4 lbs (2 kg)
Power Supply	
Width:	7⅛" (18 cm)
Height:	4" (10 cm)
Depth:	11" (28 cm)
Weight:	15 lbs (7 kg)
User Cable:	3' (1 m)



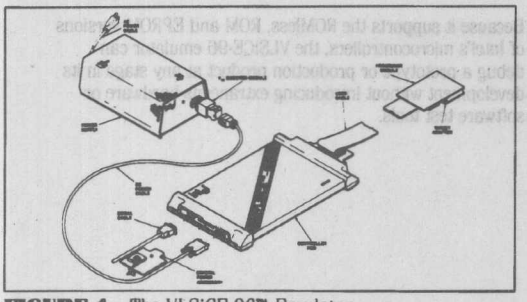


FIGURE 1. The VLSiCE-96™ Emulator

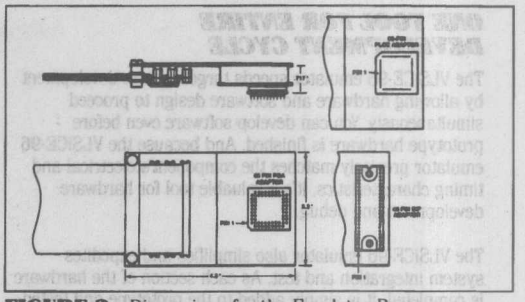


FIGURE 2. Dimensions for the Emulator Processor Board and Adaptors

Environmental Characteristics

Operating Temperature: 0°C to +40°C (−32°F to +104°F)

Operating Humidity: Maximum to 85% relative humidity, non condensing

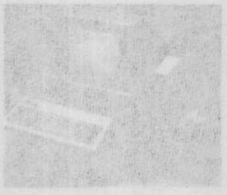
SERVICE SUPPORT AND TRAINING

Intel augments its MCS-96 architecture family development tools with a full array of seminars, classes and workshops; on-site consulting services; field application engineering expertise; telephone hot-line support; and software and hardware maintenance contracts. This full line of services will ensure your design success.

ORDERING INFORMATION

V096KITA	VLSiCE-96 Power supply cable, emulation base, user cable, Crystal Power Accessory (CPA), serial cables for PC AT/XT, a 68-pin PGA target adaptor, ASM-96, AEDIT Text Editor, Host, probe, diagnostic and tutorial software on 5¼" media for DOS hosts running DOS V3.0 or later. (Requires software license.)
V096KITD	Same as V096KITA without ASM-96 and AEDIT text editor.
TA096E	Optional 68-pin PLCC Target Adaptor Board
TA096B	Optional 48-pin DIP Target Adaptor Board
MSA96	Optional Multi-Synchronous Accessory for multi-ICE capability
SA096D	Software for host, probe, diagnostic and tutorial on 5¼" media for use with the PC AT/XT under PC-DOS V3.0 or later. (Requires software license.) (Included with V096KITA and V096KITD.)
D86C96NL	C-96 Compiler*
D86PLM96NL	PL/M-96 Compiler*
D86ASM96NL	ASM-96 Macroassembler*

*Also Includes: Relocator/Linker, object-to-hex converter, librarian, and Floating Point Arithmetic Library.



REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR

REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR

The ICE™-196KB/PC in-circuit emulator delivers real-time high-level debugging capabilities for developing, integrating and testing 80C196-based designs. Operating at the full speed of the 80C196KB microcontroller, the ICE-196KB/PC provides precise I/O pin timings and functionality. The ICE-196KB/PC also allows you to develop code before prototype hardware is available. The in-circuit emulator represents a low-cost development environment for designing real-time microcontroller-based applications with minimal investment in time and resources.

ICE™-196KB/PC IN CIRCUIT EMULATOR FEATURES

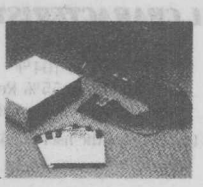
- Real-Time Emulation of the 80C196KB Microcontroller
- 64K Bytes of Mappable Memory
- 2K-entry Trace Buffer
- 3 Breakpoints or 1 Range Break
- Symbolic Support and Source Code Display
- Standalone Operation
- Versatile and Powerful Host Software
- Hosted On IBM PC XT, AT* or Compatibles With DOS 3.0 or Later

REAL-TIME EMULATION

The ICE-196KB/PC provides real-time emulation with the precise input/output pin timings and functions across the full operating frequencies of the 80C196KB microcontroller. The ICE-196KB/PC connects to the intended 80C196KB microcontroller socket via a 16″ flex cable, which terminates in a 68-pin PLCC probe. An optional 68-pin PGA adapter is also available.

MAPPABLE MEMORY

The ICE-196KB/PC has 64K bytes (65,536) of zero wait-state memory that can be enabled or mapped as read-only, write-only or read/write in 4K byte increments to simulate the internal (EP)ROM of the 80C196KB or external program memory.



intel

*PC XT, AT are trademarks of IBM.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

January, 1989

© Intel Corporation 1988

Order Number 280727-004

TRACE BUFFER

The ICE-196KB/PC contains a 2K (2048) entry trace buffer for keeping a history of actual instruction execution. The trace buffer can be conditionally turned off to collect a user-specified number of trace frames. Trace information can be displayed as disassembled instructions or, optionally, disassembled instructions and the original C-96 and PL/M-96 source code.

BREAK SPECIFICATION

Three execution address breakpoints or one range of addresses can be active at any time. The ICE-196PC allows any number of breakpoints to be defined and activated when needed.

SYMBOLIC SUPPORT AND SOURCE CODE DISPLAY

Full ASM-96, PL/M-96 and C-96 language symbolics, including variable typing and scope, are supported by the ICE-196KB/PC memory accesses, trace buffer display, breakpoint specification, and assembler/disassembler. Additionally, C-96 and PL/M-96 source code can be displayed to make development and debug easier.

STANDALONE OPERATION

Product software can be developed prior to hardware availability with the optional Crystal Power Accessory (CPA) and the ICE-196KB/PC mappable memory. The CPA also provides diagnostic testing to assure full functionality of the ICE-196KB/PC.

VERSATILE AND POWERFUL HOST SOFTWARE

The ICE-196KB/PC comes equipped with an on-line help facility, a dynamic command entry and syntax guide, built-in editor, assembler and disassembler, and the ability to customize the command set via literal definitions and debug procedures.

HOSTING

The ICE-196KB/PC is hosted on the IBM PC XT, AT or compatibles with PC-DOS 3.0 or later.

SPECIFICATIONS

REQUIREMENTS

Host

IBM PC XT, AT (or compatible)
512K bytes RAM, Hard Disk
PC-DOS 3.0 or Later
One Unused Peripheral Slot
DC Current 2.5A
ICE-196KB/PC 2 Bytes of User Stack Space

TARGET INTERFACE BOARD

Length 2.0" (5.1cm)
Height 1.2" (3.0cm)
Width 2.3" (5.8cm)

USER CABLE

Length 15.6" (39.6cm)

PROBE ELECTRICAL

80C196KB plus per pin	50pf loading 5ns propagation delay
Icc (from target system)	50mA @ 12 MHz
Operating Frequency	3.5 to 12 MHz. 12 MHz only with CPA

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature	10°C to 40°C 37.5°F to 104°F
Operating Humidity	Maximum 55% Relative Humidity, non-condensing

Note: ICE-196KB/PC uses two bytes of the user stack.

ORDERING INFORMATION

Order Code	Description
ICE196KBPC	Emulation Board, user cable, target interface board (PLCC), host, diagnostic, and tutorial software on 5¼" DOS diskette, and Crystal Power Accessory with power cable
ICE196KBPCB	Same as above except does not include Crystal Power Accessory
TA196PLCC68PGA	68 pin PGA target adapter
CPA196KAKB	Crystal Power Accessory and power cable only
D86C96NL	C-96 Compiler*
D86PLM96NL	PL/M-96 Compiler*
D86ASM96NL	ASM-96 Assembler*

*Includes: Relocator/Linker, Object-to-hex Converter, Floating Point Arithmetic Library, Librarian

ICE™-196KB/xX IN-CIRCUIT EMULATORS

MODULAR IN-CIRCUIT EMULATORS FOR THE 8xC196KB FAMILY OF MICROCONTROLLERS

The ICE™-196KB/MX and ICE-196KB/HX in-circuit emulators deliver a complete, real-time, hardware/software debug environment for developing, integrating, and testing 8xC196KB-based designs. The ICE-196KB/MX emulator is a mid-range modular debugging system featuring high performance 12 MHz emulation, high-level symbolic debugging, 64k bytes zero-waitstate mappable memory, and emulation trace. ICE-196KB/HX emulator is a high-end system with all the functionality of ICE-196KB/MX plus additional break/trace capabilities and expanded mappable memory. The ICE-196KB/MX emulator can be upgraded to an ICE-196KB/HX emulator with optional add-in boards. Both systems feature an identical human interface, utilize the same base chassis, and are serially hosted on IBM* PC XTs and ATs, and 100% compatibles.

ICE-196KB/xX IN-CIRCUIT EMULATORS CORE FEATURES

- **Precisely** matches the component's electrical and timing characteristics
- Supports the ROMless and (EP)ROM versions of the 8xC196KB
- Does not introduce extraneous hardware or software overhead
- Modular base for future growth and migration

intel

*IBM is a registered trademark of International Business Machines.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1988

October, 1988
Order Number: 280847-001

ICE™-196KB/MX IN-CIRCUIT EMULATOR FEATURES

- Real-time transparent emulation of the 8xC196KB microcontroller family up to 12 MHz, including ROM and EPROM versions
- 64k bytes of zero-waitstate mappable memory to allow early software debug and (EP)ROM simulation, expandable to 128k bytes
- 64k hardware execution breakpoints
- Symbolic debugging and source code display for faster and easier access to memory location and program variables
- 2k frame trace buffer displaying execution address
- Multi-ICE synchronization to start and stop multiple emulators in multi-processor designs
- Run-time viewable execution trace
- Watch window feature automatically displays variables when breaking emulation
- Serially-hosted (RS232C) with very high-speed download capability
- ONCE™ support for on-circuit emulation of surface-mount target systems
- Trigger out for synchronization with external logic analyzer or scope
- Capable of suspension mounting for remote debug
- Full language support with ASM-96, PL/M-96, and C-96
- On-line disassembler and single-line assembler
- Context-sensitive drop-down "help" window to speed development
- On-line tutorial
- Self-test diagnostics to ensure system integrity
- World-wide service and support

ICE™-196KB/HX IN-CIRCUIT EMULATOR FEATURES

Includes all features in ICE™-196KB/MX emulator *plus the following:*

- Additional complex event recognizers for bus break/trace to allow debugging on data values, events, or addresses
- Dynamic trace allows user to view trace buffer without stopping emulation
- Fastbreaks to allow the user to access program variables and SFRs during emulation
- Additional 64k bytes of zero-waitstate mapped memory (128k bytes total)
- Emulation timer and event timer for debugging speed-critical applications and to allow performance analysis capabilities
- Conditional trace to allow tracing under user-specified conditions
- Asynchronous external break capability
- Full multi-ICE communication for enhanced debugging in multi-processor designs
- Input and output logic clips for external logic analysis and control
- 20k bytes additional trace buffer displaying execution address, bus address, bus data, bus status, and clips in
- Run-time reprogrammable break/trace points

COMPLETE FAMILY OF 8x196 DEVELOPMENT TOOLS

ICE-196KB/MX and ICE-196KB/HX emulators are complemented by Intel's low-cost ICE-196KB/PC emulator. All three emulators utilize an upward-compatible human interface to preserve your learning investment and to allow multiple emulators for large design teams. Each emulator has been designed to work in conjunction with Intel's MCS-96 software tools, including a macroassembler, a PL/M-96 compiler, a C-96 compiler, and various utilities.

Optional boards are available to upgrade an ICE-196KB/MX emulator with some or all of the functionality of an ICE-196KB/HX emulator. In addition, the ICE-196KB/MX and ICE-196KB/HX emulators have been designed to support future proliferations within the 8xC196 family of microcontrollers.

WORLDWIDE SERVICE AND SUPPORT

Intel augments its MCS-96 architecture family development tools with a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hot-line support; and software and hardware maintenance contracts. This full line of services will ensure your design success.

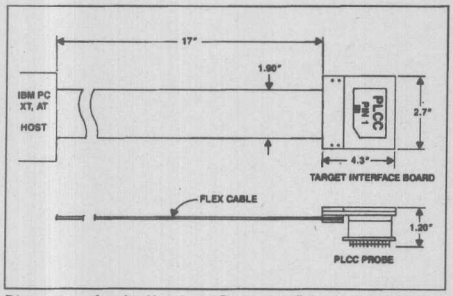
SPECIFICATIONS

HOST REQUIREMENTS

Emulators require an IBM PC AT/XT (or 100% compatible) with 512k bytes RAM and hard disk running DOS 3.1 or higher. Intel recommends 640k bytes of RAM.

ELECTRICAL CHARACTERISTICS

Power Supply: 100V-120V or 200V-240V
50 Hz–60 Hz
5 amps (AC max) @ 120V
2 amps (AC max) @ 240V



Dimensions for the Emulator Processor Board and Adapters

PHYSICAL CHARACTERISTICS

Target Probe

Width: 6.9 cm (2.7")
Height: 3.0 cm (1.2")
Length: 11.0 cm (4.3")
Package: 68-pin PLCC (optional 68-pin PGA flexible adapter available)

Emulator Chassis

Width: 34 cm (13 3/8")
Height: 12 cm (4 1/2")
Depth: 25 cm (9 7/8")
Weight: 3.2 kg (7 lb)

Power Supply

Width: 18 cm (7 1/2")
Height: 10 cm (4")
Depth: 28 cm (11")
Weight: 7 kg (15 lb)

Probe Cable Length: 40 cm (17")
Serial Cable Length: 3.65 m (12')

ENVIRONMENTAL CHARACTERISTICS

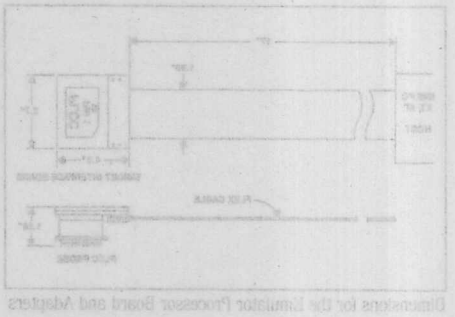
Operating Temperature: 0°C to 40°C
Operating Humidity: Maximum 85% relative humidity, non-condensing

ORDERING INFORMATION

ICE196KBHX	ICE in-circuit emulator base chassis, 196 emulation control board (ECB), 196KB target probe, 196KB crystal power accessory (CPA), enhanced break/trace board (BTB), 64k optional memory board (OMB), clips in/out, power supply and cable, serial cables for PC XT/AT, 68-pin PLCC target adapter. Host, 196KB probe, diagnostic, and tutorial software on 5¼" media for DOS hosts running DOS 3.1 or later. (Requires software license.)
ICE196KBMX	Same as ICE196KBHX except without enhanced break/trace board (BTB), without 64k optional memory board (OMB), and without clips in/out
ICEBTB	Enhanced break/trace board (BTB) for upgrading an ICE-196KB/MX system
ICEOMB	Optional memory board with 64k zero-waitstate mapped memory for upgrading an ICE-196KB/MX system
ICECLIPS	Clips in/out for upgrading an ICE-196KB/MX system (requires an enhanced break/trace board)
TA196PLCC68PGA	68-pin PGA target adapter
ICEXONCE	Target adapter for ONCE (on-circuit) emulation
ICE196KBPC	ICE-196KB/PC—PC form-factor in-circuit emulator
D86ASM96NL	ASM-96 macroassembler*
D86C96NL	C-96 compiler*
D86PLM96NL	PL/M-96 compiler*

*Also includes: Relocator/Linker, object-to-hex converter, librarian, and floating point arithmetic library

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).



Using the 80186/188/C186/C188

8

801896\188\C1896\C188

USING THE 80186/188/C186/C188

1.0 INTRODUCTION

The 80186 microprocessor family holds the position of industry standard among high integration microprocessors. VLSI technology incorporates the most commonly used peripheral functions with a 16-bit CPU on the same silicon die to assure compatibility and high reliability (see Figure 1). The 80186 reputation for flexibility and uncomplicated programming make it the first choice microprocessor for such data control applications as local area network equipment, PC add-on cards, terminals, disk storage subsystems, avionics, and medical instrumentation.

There are two purposes to this Application Note. The first is to explain the operation of the integrated 80186 peripheral set with a degree of detail not possible in the data sheet. The second is to describe, through examples, the use of the 80186 with other digital logic such as memory.

The 80186 family actually consists of 4 devices: the original 80186 and 80188, and the new 80C186 and 80C188 microprocessors manufactured on Intel's CHMOS III process. The 80188 and 80C188 are 16-bit microprocessors but have 8-bit external data buses. The 80C186 and 80C188 offer the advantage of increased speed (up to 16 MHz) and important new features including a Refresh Control Unit, Power-Save Logic, and ONCE™ Mode (see Figure 2). For simplicity, this Ap Note uses the name 80186 to refer collectively to all the members of the 80186 family. Differences between individual processors are pointed out as necessary.

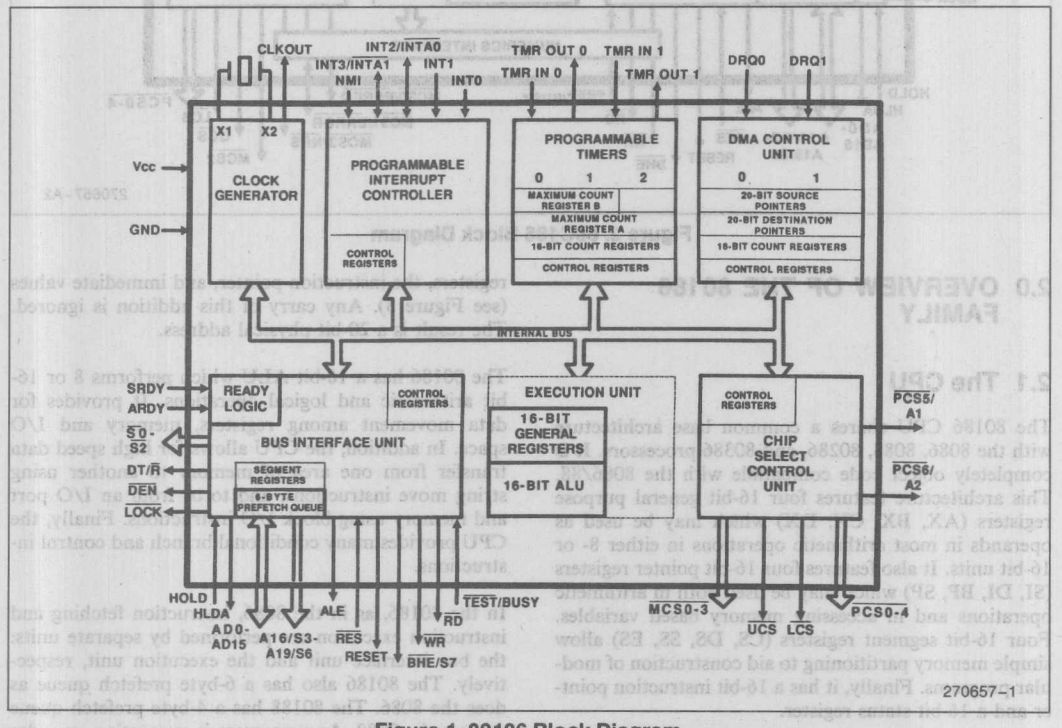


Figure 1. 80186 Block Diagram

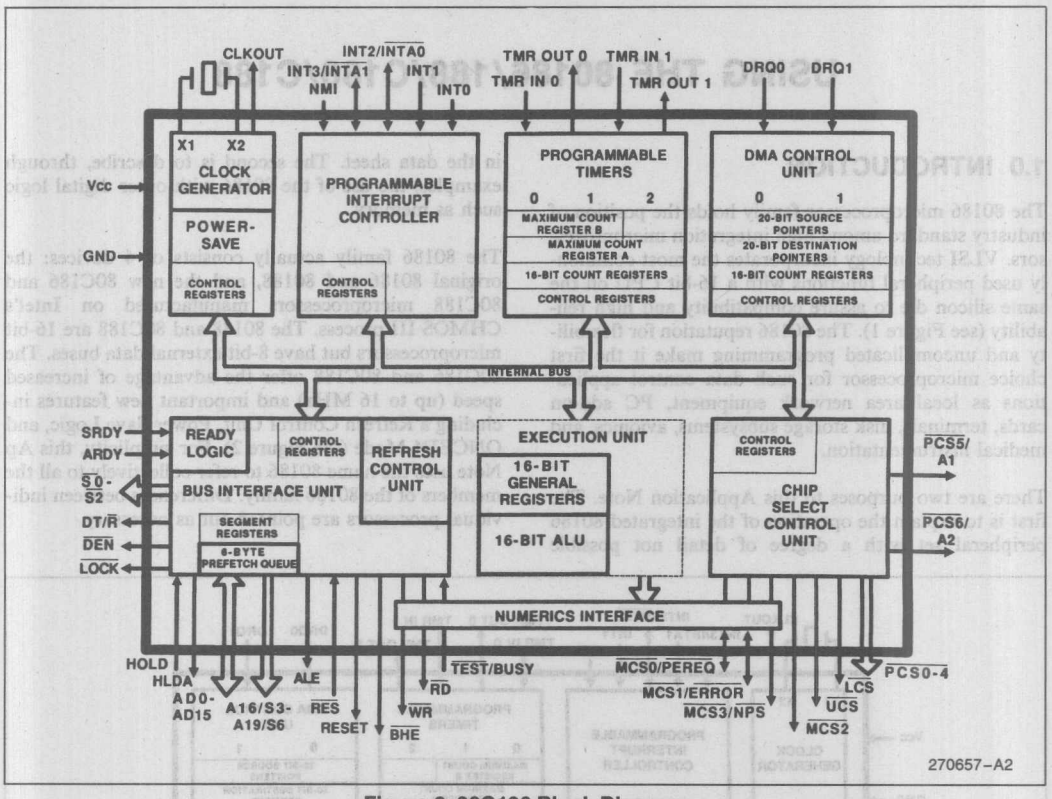


Figure 2. 80C186 Block Diagram

2.0 OVERVIEW OF THE 80186 FAMILY

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088, 80286, and 80386 processors. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX, BX, CX, DX) which may be used as operands in most arithmetic operations in either 8- or 16-bit units. It also features four 16-bit pointer registers (SI, DI, BP, SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS, DS, SS, ES) allow simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is shifted left 4 bits and then added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 3). Any carry of this addition is ignored. The result is a 20-bit physical address.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides many conditional branch and control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is excecuting, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another opcode byte, it takes the byte out of the queue. Effective processor throughput is increased by

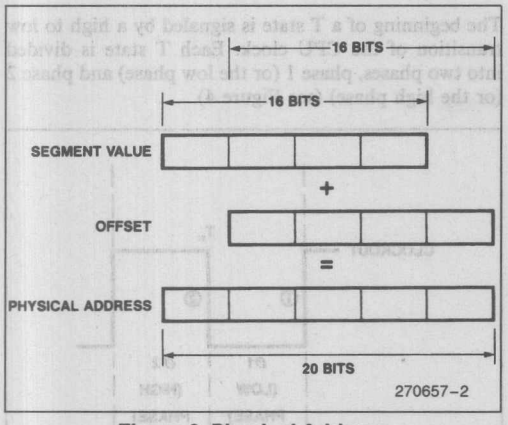


Figure 3. Physical Address Generation in the 80186

adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 has many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce code size. The added instructions are described in Appendix H of this Ap Note.

2.3 DMA Unit

The 80186 includes a DMA unit which provides two flexible DMA channels. This DMA unit will perform transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations.

Each DMA channel maintains independent 20-bit source and destination pointers. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be optionally incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which can terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The timer unit contains 3 independent 16-bit timer/counters. Two of them can count external events, provide waveforms based on either the CPU clock or an external clock, or interrupt the CPU after a specified count. The third timer/counter counts only CPU clocks. After a programmable interval, it can interrupt the CPU, provide a clock pulse to either or both of the other timer/counters, or send a DMA request pulse to the integrated DMA controller.

2.5 Interrupt Controller

The integrated interrupt controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to an external 8259A or 82C59A interrupt controller. In addition, it can be configured as a slave controller.

2.6 Clock Generator

The on-board crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing is derived. The CPU clock is externally available, and all timing parameters are referenced to it.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 80186 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in

either memory or I/O space so that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These bits allow a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, a bit determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0–3 and peripheral devices 4–6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals.

3.0 USING THE 80186 FAMILY

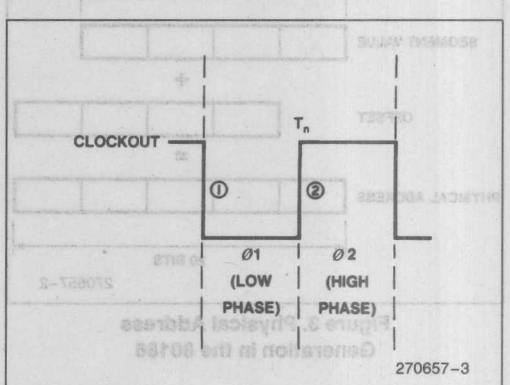
3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to that of the 8086. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles intitiated by the 80186 integrated DMA unit.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1, T_2, T_3, T_W and T_4. Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number of wait states (t_W) inserted in each bus cycle. The maximum number of wait states is unbounded.

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) (see Figure 4).



NOTES:
1. Falling edge of T_n.
2. Rising edge of T_n.

Figure 4. T-state in the 80186

Different types of bus activity occur for all of the T-states (see Figure 5). Address generation information occurs during T_1, data generation during T_2, T_3, T_W and T_4. The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Information concerning an impending bus cycle appears during the T-state immediately before the first T-state of the cycle itself. Two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediately followed by an idle T state.

During the first type of T_4 or T_i, status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i, the status outputs remain inactive because no bus cycle will follow. The decision on which type T_4 or T_i state to present is made at the beginning of the T-state preceding the T_4 or T_i state (see Figure 6). This determination has an effect on bus latency (see Section 3.3.2).

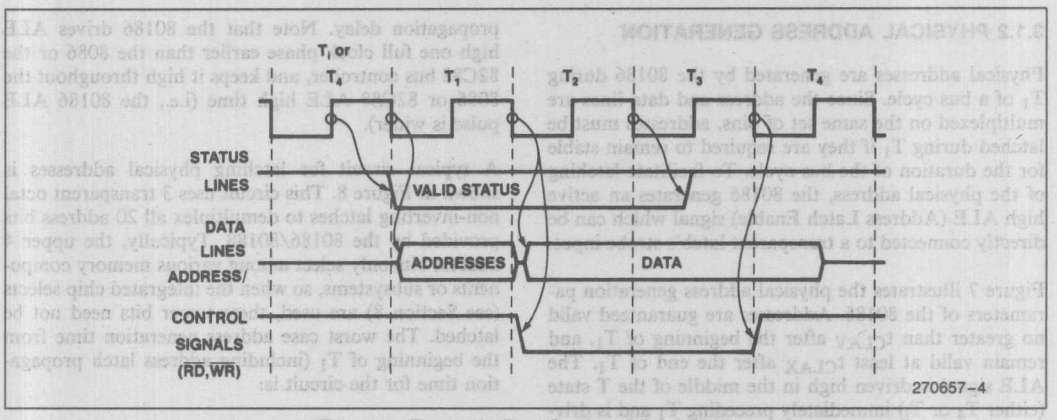


Figure 5. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0–AD15
address/status	A16/S3-A19-S6, $\overline{BHE}$/S7
co-processor control	$\overline{TEST}$
local bus arbitration	HOLD, HLDA
local bus control	ALE, $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, $\overline{DEN}$
multi-master bus	$\overline{LOCK}$
ready (wait) interface	SRDY, ARDY
status information	$\overline{S0}$–$\overline{S2}$

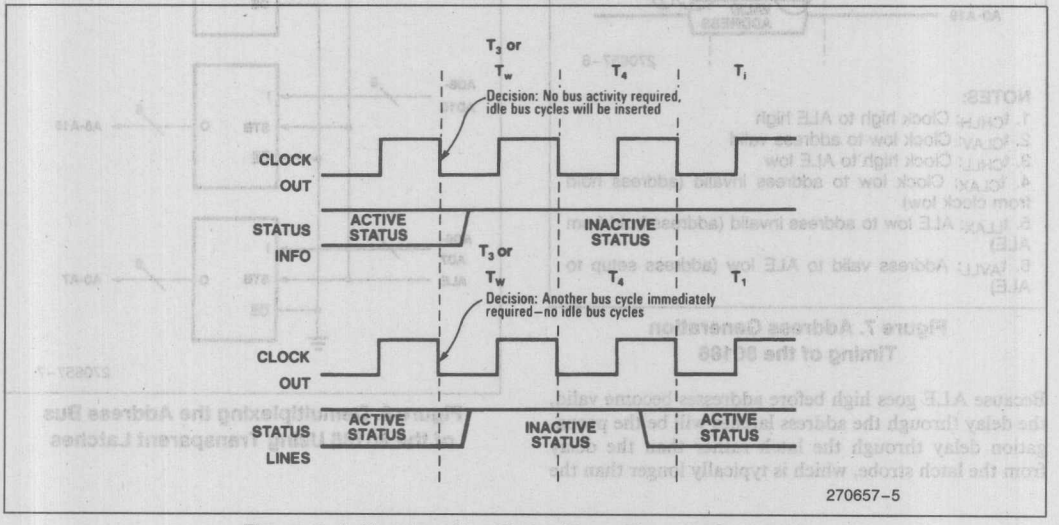
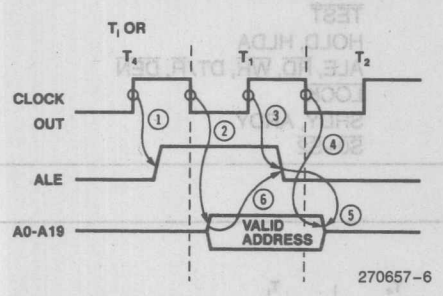


Figure 6. Active-Inactive Status Transitions in the 80186

3.1.2 PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 7 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater than t_{CLAV} after the beginning of T_1, and remain valid at least t_{CLAX} after the end of T_1. The ALE signal is driven high in the middle of the T state (either T_4 or T_i) immediately preceding T_1 and is driven low in the middle of T_1, no sooner than t_{AVLL} after addresses become valid. This parameter (t_{AVLL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} after ALE goes inactive to satisfy address latch hold times.



NOTES:
1. t_{CHLH}: Clock high to ALE high
2. t_{CLAV}: Clock low to address valid
3. t_{CHLL}: Clock high to ALE low
4. t_{CLAX}: Clock low to address invalid (address hold from clock low)
5. t_{LLAX}: ALE low to address invalid (address hold from ALE)
6. t_{AVLL}: Address valid to ALE low (address setup to ALE)

Figure 7. Address Generation Timing of the 80186

Because ALE goes high before addresses become valid, the delay through the address latches will be the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 82C88 bus controller, and keeps it high throughout the 8086 or 82C88 ALE high time (i.e., the 80186 ALE pulse is wider).

A typical circuit for latching physical addresses is shown in Figure 8. This circuit uses 3 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 80186/80188. Typically, the upper 4 address bits only select among various memory components or subsystems, so when the integrated chip selects (see Section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time for the circuit is:

$$t_{CLAV} + t_{PD}$$

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

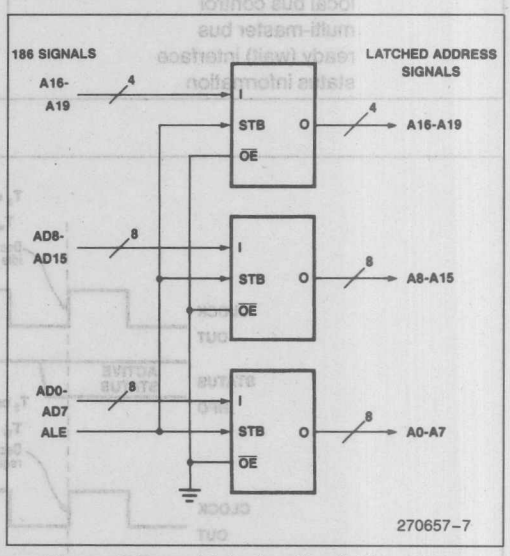


Figure 8. Demultiplexing the Address Bus of the 80186 Using Transparent Latches

One more signal is generated by the 80186 to address memory: $\overline{BHE}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus. Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1–A19, not A0–A19. This provides 512K unique **word** addresses, or 1M unique **byte** addresses. $\overline{BHE}$ is not present on the 8-bit 80188. All data transfers occur on the 8-bits of the data bus.

3.1.3 80186/80C186 DATA BUS OPERATION

Throughout T_2, T_3, T_W and T_4 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either bytes or words. All memory is byte addressable (see Figure 9).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{BHE}$ is driven high, and the data transfer occurs on D0–D7 of the data bus. Whenever an access is made to only the odd byte, $\overline{BHE}$ is driven low, A0 is driven high, and the data transfer occurs on D8–D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{BHE}$ are driven low and the data transfer occurs on D0–D15 of the databus.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8–D15, the second to access the even byte at the next sequential word address on D0–D7. For example, in Figure 9, byte 0 and byte 1 can be individually accessed in two separate bus cycles to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the unused half of the data bus is ignored. When byte writes are made, the data driven on the unused half of the data bus is indeterminate.

3.1.4 80188/80C188 DATA BUS OPERATION

Because the 80188 and 80C188 externally have only 8-bit data buses, the above discussion about upper and lower bytes of the data bus does not apply. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access to lower byte of the word; the second to access the upper byte of the word.

Any 80188/80C188 access to the integrated peripherals is performed 16 bits at a time, whether byte or word addressing is used. If a byte operation is used, the external bus only indicates a single byte transfer even though the word access takes place.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186, additional buffering may not be required in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it float its output drivers after a read **before** the 80186 begins to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until addresses active for the next bus cycle (t_{RHAV}). If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the device from driving these lines concurrently. This parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

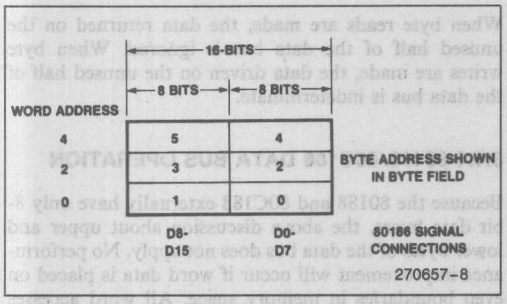


Figure 9. Physical Memory Byte/Word Addressing in the 80186

If data buffers are required, the 80186 provides $\overline{DEN}$ (Data ENable) and DT/$\overline{R}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{DEN}$ and DT/$\overline{R}$ signals are activated during all bus cycles. The $\overline{DEN}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write). In other words, $\overline{DEN}$ is low during any active bus cycle when address information is not being generated on the address/data pins. In most systems, the $\overline{DEN}$ signal should **not** be directly connected to the $\overline{OE}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{DEN}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable.

The DT/$\overline{R}$ signal determines the direction of data through the bi-directional buffers. It is high whenever data is being written from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{DEN}$ signal, it may be directly connected to bus buffers, since this signal does not usually enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 10. Note that the A side of the buffer is connected to the 80186, the B side to the external device. The DT/$\overline{R}$ signal can directly drive the T (transmit) signal of a typical buffer since it has the correct polarity.

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals $\overline{RD}$, $\overline{WR}$, $\overline{LOCK}$ and $\overline{TEST}$. In addition, the 80186 provides the status signals S0–S2 and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{RD}$ and $\overline{WR}$

The $\overline{RD}$ and $\overline{WR}$ signals strobe data to or from memory or I/O space. The $\overline{RD}$ signal is driven low at the beginning of T_2, and is driven high at the beginning of T_4 during all memory and I/O reads (see Figure 11). $\overline{RD}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4. $\overline{RD}$ will not go inactive until the processor's data hold time has been satisfied.

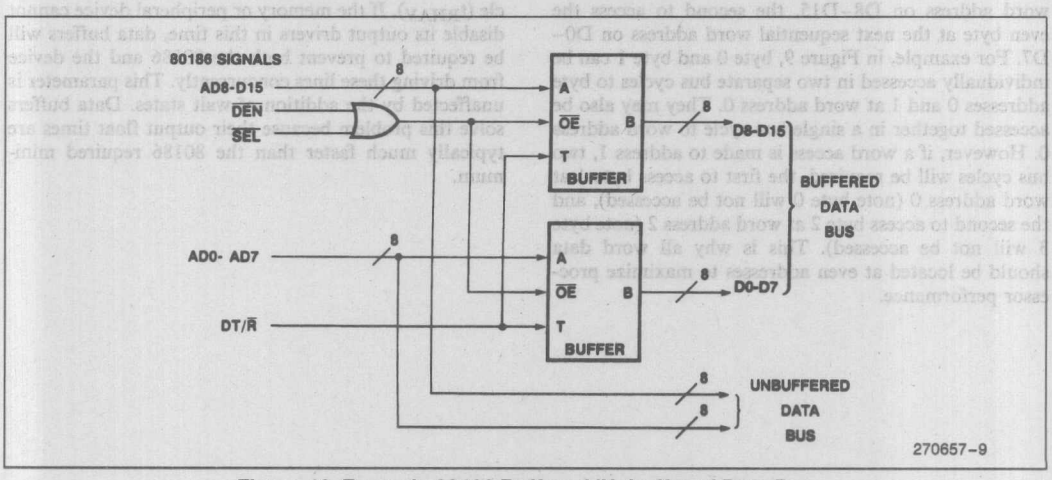
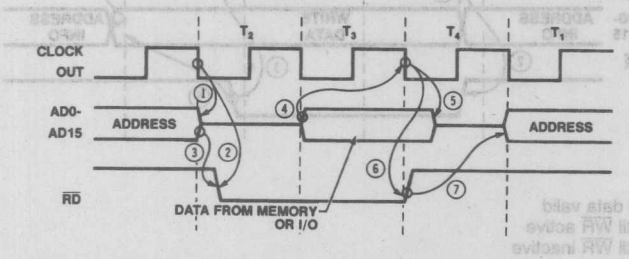


Figure 10. Example 80186 Buffered/Unbuffered Data Bus

Note that the 80186 does not provide separate I/O and memory $\overline{RD}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{RD}$ signal (see Figure 12). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to an inactive state well before the beginning of T_4 (where $\overline{RD}$ goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ goes to the inactive state (high). The status signals may be latched using ALE the same as the address signals (often using the spare bits in the address latches).

Often the lack of a separate I/O and memory RD signal is not important in an 80186 system. Each 80186 chip select signal will respond to accesses exclusively in memory or I/O space. Thus, when a chip select is used, the external device is enabled only during accesses to the proper address in the proper space.

The $\overline{WR}$ signal is also driven low at the beginning of T_2 and driven high at the beginning of T_4 (see Figure 13). In similar fashion to the $\overline{RD}$ signal, the $\overline{WR}$ signal is active for all memory and I/O writes. Again, separate memory and I/O control lines may be generated using the latched $\overline{S2}$ signal along with $\overline{WR}$. More important, however, is the role of the active-going edge of $\overline{WR}$. At the time $\overline{WR}$ makes its high-to-low transition, valid write data is **not** present on the data bus. This has consequences when using $\overline{WR}$ to enable such devices as DRAMs since those devices require the data to be stable on the falling edge. In DRAM applications, the problem is solved by the DRAM controller (an Intel 8207, for example). For other applications which require valid data before the $\overline{WR}$ transition, place cross-coupled NAND gates between the CPU and the device on the $\overline{WR}$ line (see Figure 14). The added gates delay the active-going edge of $\overline{WR}$ to the device by one clock phase, at which time valid data is driven on the bus by the 80186.



NOTES:
1. t_{CLAZ}: Clock low until address float
2. t_{CLRL}: Clock low until $\overline{RD}$ active
3. t_{AZRL}: Address float until $\overline{RD}$ active
4. t_{DVCL}: Data valid until clock low (data input set-up time)
5. t_{CLDX}: Clock low unitl data invalid (data input hold time from clock)
6. t_{CLRH}: Clock low until $\overline{RD}$ high
7. t_{RHAV}: $\overline{RD}$ high until addresses valid

Figure 11. Read Cycle Timing of the 80186

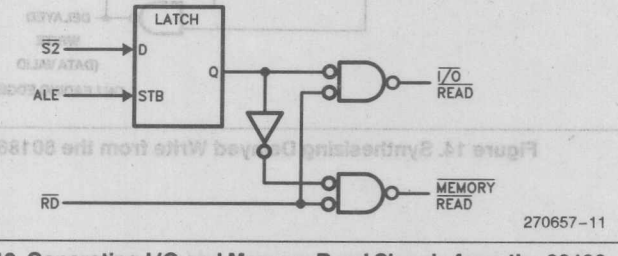


Figure 12. Generating I/O and Memory Read Signals from the 80186

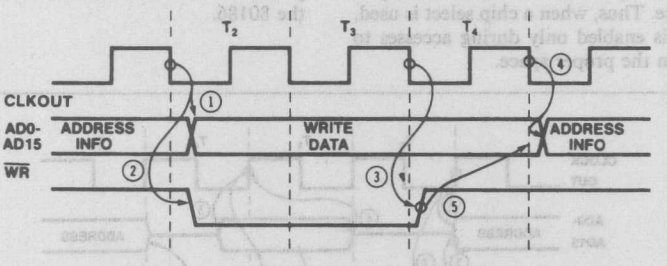
3.1.6.2 Queue Status Signals

If the $\overline{RD}$ line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter Queue Status Mode. When in this mode, the $\overline{WR}$ and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 ($\overline{WR}$) is given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines $\overline{S0}$–$\overline{S2}$ using an external 82C88 bus controller (see Figure 15). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device.



NOTES:
1. t_{CLDV}: Clock low until data valid
2. t_{CVCTV}: Clock low until $\overline{WR}$ active
3. t_{CVCTX}: Clock low until $\overline{WR}$ inactive
4. t_{CLDOX}: Clock high until data invalid
5. t_{WHDX}: $\overline{WR}$ inactive until data invalid

Figure 13. Write Cycle Timing of the 80186

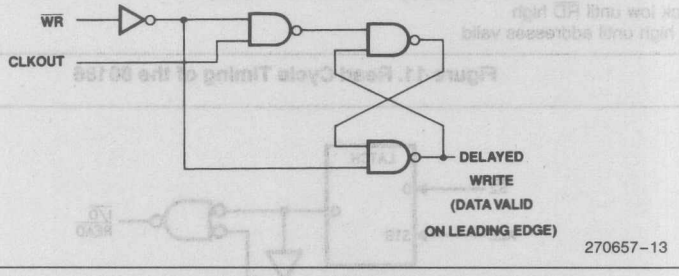


Figure 14. Synthesizing Delayed Write from the 80186

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 6). The possible status line encodings are given in Table 3. The status lines are driven to their inactive state in the T_3 or T_W state immediately preceding T_4 of the current bus cycle.

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The status lines may be directly connected to an 82C88 bus controller, which provides local bus control signals or multi-bus control signals (see Figure 15). Use of the 82C88 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals can provide local bus control signals, while an 82C88 can provide multibus control signals.

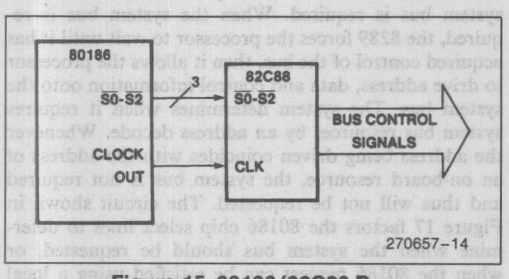


Figure 15. 80186/82C88 Bus Controller Interconnection

Two additional status signals are provided by 80186 family members. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2, T_3, T_4 and T_W. In the 8086 family, all central processors (e.g., the 8086 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transfering data for the DMA unit.

S7 and $\overline{BHE}$ are logically equivalent signals provided by the 80186 and the 80C186 (see Section 3.1.2). S7 is always high on the 80188 and 80C188 (except during 80C188 DRAM refresh cycles) which signifies the presence of an 8-bit data bus.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address has been derived. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will always be low.

3.1.6.4 $\overline{TEST}$ and $\overline{LOCK}$

Finally, the 80186 provides a $\overline{TEST}$ input and a $\overline{LOCK}$ output. The $\overline{TEST}$ input is used in conjunction with the processor WAIT instruction. It is typically driven by a coprocessor to indicate whether it is busy.

The $\overline{LOCK}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. The $\overline{LOCK}$ signal indicates to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are typically used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources.

On the 80186, the $\overline{LOCK}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive during T_4 of the last DATA cycle of the locked transfers (assuming no wait states). On the 8086, the $\overline{LOCK}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{LOCK}$ signal before the first LOCKED data cycle is performed. Since $\overline{LOCK}$ is active before the 8086 requires the bus for the data transfer, opcode pre-fetching can be LOCKED. LOCKED prefetching will not occur with the 80186.

The LOCK output is also driven low during interrupt acknowledge cycles when the integrated interrupt controller operates in Cascade or Slave Modes (see Sections 6.5.2 and 6.5.3). In these modes, the operation of the LOCK pin may be altered when an interrupt occurs

during execution of a software-LOCKED instruction. See Section 6.5.4 for a description of additional hardware necessary to block DMA and HOLD requests under such circumstances.

3.1.7 HALT TIMING

A HALT bus cycle signifies that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two ways.

The first way a HALT bus cycle differs from a normal bus cycle is that neither $\overline{RD}$ nor $\overline{WR}$ will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{S0}$–$\overline{S2}$ status lines go to their inactive state (all high) during T_2 of the bus cycle, well before they go to their inactive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{S0}$–$\overline{S2}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 (see section 4.4.1).

3.1.8 82C88 AND 8289 INTERFACING

The 82C88 and 8289 are the bus controller and multimaster bus arbitration devices used with the 8086. Because the 80186 bus is similar to the 8086 bus, they can be used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 82C88 bus controller generates control signals ($\overline{RD}$, $\overline{WR}$, ALE, DT/$\overline{R}$, $\overline{DEN}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{S0}$–$\overline{S2}$ of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 82C88 just as in an 8086 system. Using the 82C88 with the 80186 does not prevent using the 80186 control signals. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the 82C88 lines used as the system signals. Note that in an 80186 system, the 82C88 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 82C88 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

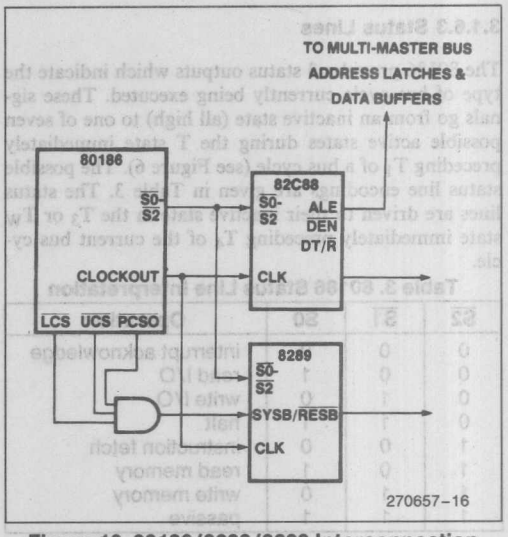
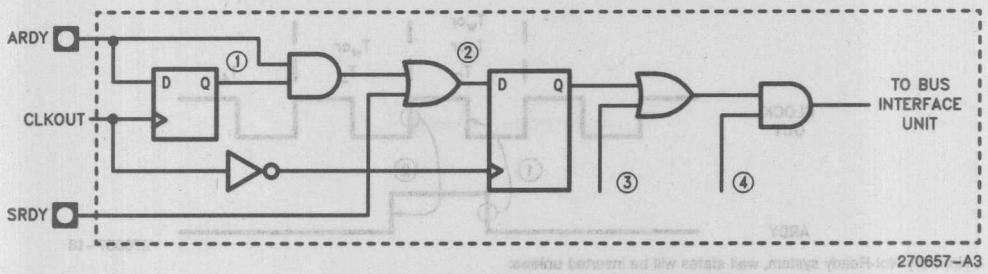


Figure 16. 80186/8288/8289 Interconnection

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices, each of which can become the bus master. This component also decodes status lines $\overline{S0}$–$\overline{S2}$ directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown in Figure 17 factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the bus controller to insert wait states (T_W) into a CPU bus cycle, allowing slower devices to respond to bus activity. Wait states will only be inserted when both ARDY and SRDY are low, i.e., **only one** of the lines need be active to terminate a bus cycle. Figure 17 depicts the logical ORing of the ARDY and SRDY functions. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers and to any area where the chip select ready bits indicate that the external ready should be ignored.



NOTES:
1. Asynchronous Resolution Flip Flop
2. Ready Latch Flip Flop
3. R2 Bit in Chip Select Registers (Selects Internal Only (1) or External (0))
4. Internally Generated Unit States Controlled by R0 and R1 Bits in Chip Select Registers

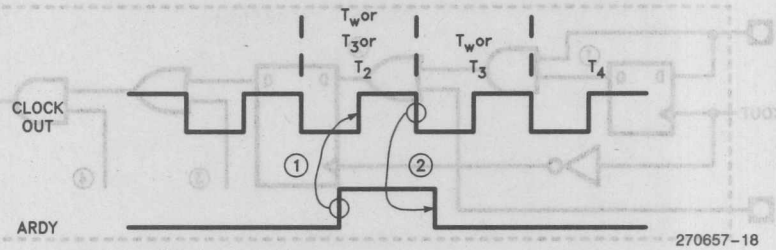
Figure 17. 80186 Ready Circuitry

The timing required by the two RDY lines is different. Inputs to the ARDY pin will be internally synchronized to the CPU clock before being presented to the rest of the bus control logic as shown in Figure 17. The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite high or low level before its output is latched into the second flip-flop. When latched high, it passes along the level present on the ARDY line; when latched low, it forces not ready to be passed along to the rest of the circuit. (See Appendix B for synchronizer information.)

Figure 18 depicts activity for Normally-Ready and Normally-Not-Ready configurations of external logic. Remember that for ARDY to force wait states, SRDY must be low as well.

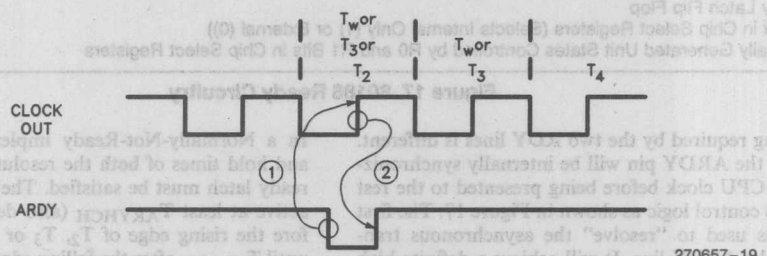
In a Normally-Not-Ready implementation the setup and hold times of **both** the resolution flip-flop **and** the ready latch must be satisfied. The ARDY pin must go active at least T_{ARYHCH} (also denoted T_{ARYCH}) before the rising edge of T_2, T_3 or T_W, and stay active until T_{CLARX} after the falling edge of T_3 or T_W to stop generation of wait states and terminate the bus cycle. If ARDY goes active before the rising edge of T_2 and stays active after the falling edge of T_3 there will be no wait state inserted.

In a Normally-Ready implementation the setup and hold times of **either** the resolution flip-flop **or** the ready latch must be met. Wait states will be generated if ARDY goes inactive T_{ARYHCH} (also denoted



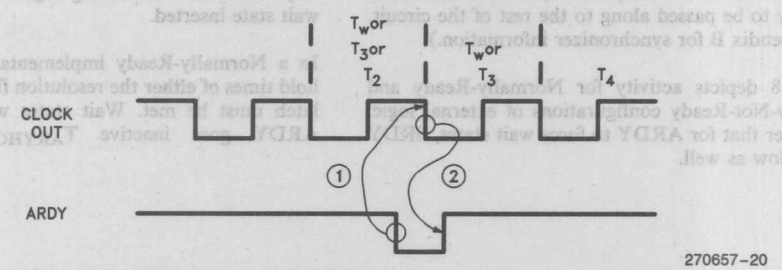
270657-18

In a Normally-Not-Ready system, wait states will be inserted **unless**:
1. t_{ARYHCH} (also denoted t_{ARYCH}): ARDY active to clock high (ARDY resolution setup time)
2. t_{CLARX}: Clock low to ARDY inactive (ARDY active hold time)



270657-19

In a Normally-Ready system, wait states will be inserted **if**:
1. t_{ARYHCH} (also denoted t_{ARYCH}): ARDY low to clock high (ARDY resolution setup time)
2. t_{ARYCHL}: Clock high to ARDY high (ARDY inactive hold time)



270657-20

Alternatively, in a Normally-Ready system, wait states will be inserted **if**:
1. t_{ARYLCL}: ARDY low to clock low (ARDY setup time)
2. t_{CLARX}: Clock low to ARDY high (ARDY active hold time)

ARDY must meet T_{ARYLCL} and T_{CLARX} or undesired CPU operation will result.

Figure 18. ARDY Transitions

T_{ARYCH}) before the rising edge of T_2 and stays inactive a minimum of T_{ARYCHL} after the edge, or if ARDY goes inactive at least T_{ARYLCL} before the falling edge of T_3 and stays inactive a minimum of T_{CLARX} after the edge. The 80186 ready circuitry performs this way to allow a slow device the maximum amount of time to respond with a not ready after it has been selected.

The synchronous ready (SRDY) line requires that **all** transitions on this line during T_2, T_3, or T_W satisfy setup and hold times (t_{SRYCL} and t_{CLSRY} respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line and subsequent wait state insertion is shown in Figure 19 the bus controller looks at SRDY at the beginning of each T_3 and T_W. If the line is sampled active at the beginning of either of these two cycles, that cycle will be immediately followed by T_4. If the line is sampled inactive at the beginning of either T state, that cycle will be followed by a T_W. Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_W, i.e., when the processor is not sampling the input, will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. The reader should recall that a separate unit, the bus interface unit, fetches opcodes from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

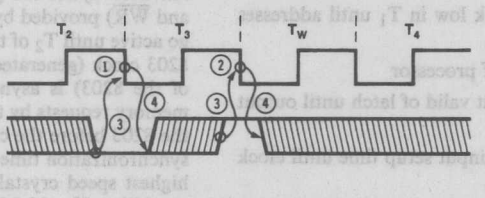
If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions (for example, data move instructions), the execution will be bus limited. Here, the execution unit will have to wait often for an instruction to be fetched before it continues. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

Although the amount of bus utilization will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. The 80186 executes most instructions in fewer clock cycles, thus requiring instructions from the queue at a faster rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest is the example EPROM interface shown in Figure 20.



NOTES:
1. Decision: Not Ready, T-State will be followed by a wait state
2. Decision: Ready, T-State will **not** be followed by a wait state
3. t_{SRYCL}: Synchronous ready stable until clock low (SRDY set-up time)
4. t_{CLSRY}: Clock low until synchronous ready transition (SRDY hold time)

Figure 19. Valid SRDY Transitions

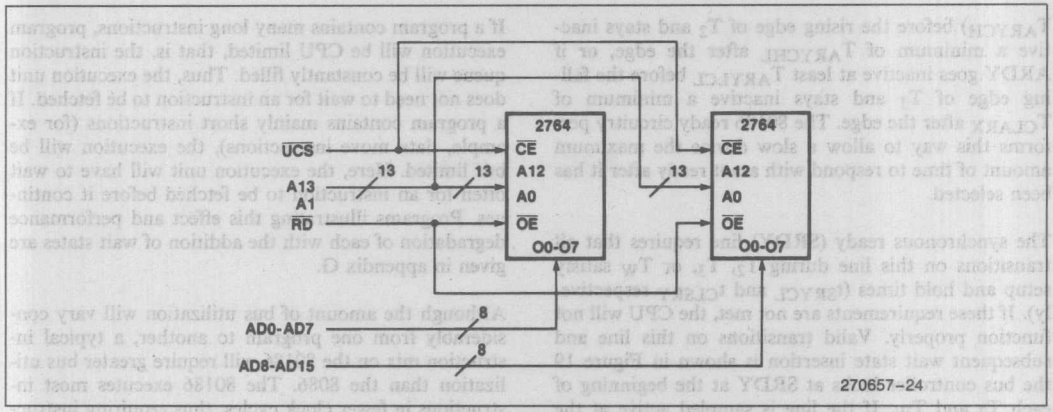


Figure 20. Example 2764/80186 Interface

The addresses are latched using the circuit shown earlier. Note that the A0 line of each EPROM is connected to the A1 address line from the 80186, NOT the A0 line. Remember, A0 only signals a data transfer on the lower 8 bits of the 16-bit data bus. The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 $\overline{RD}$ signal is used as the $\overline{OE}$ for the EPROMs.

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

time from address: $(3 + N) * t_{CLCL} - t_{CLAV} - t_{PD}$ (latch) $- t_{DVCL}$

time from chip select: $(3 + N) * t_{CLCL} - t_{CLCSV} - t_{DVCL}$

time from $\overline{RD}$ ($\overline{OE}$): $(2 + N)\ t_{CLCL} - t_{CLRL} - t_{DVCL}$

where:

t_{CLAV} = time from clock low in T_1 until addresses are valid

t_{CLCL} = clock period of processor

t_{PD} = time from input valid of latch until output valid of latch

t_{DVCL} = 186 data valid input setup time until clock low time in T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{RD}$ goes low

N = number of wait states inserted

The only significant parameter not included above is t_{RHAV}, the time from $\overline{RD}$ inactive (high) until the 80186 begins driving address information. The output float time of the EPROM must be within this spec. If slower EPROMs are used, a discrete buffer **must** be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 21. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface to 64K DRAMs.

All 8203 cycles are generated off control signals ($\overline{RD}$ and $\overline{WR}$) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatability. If a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150 ns DRAMs with an 8 MHz 80186, and three wait states if 200 ns DRAMs are used (see Figure 22).

The entire DRAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also insert the wait states required by the interface.

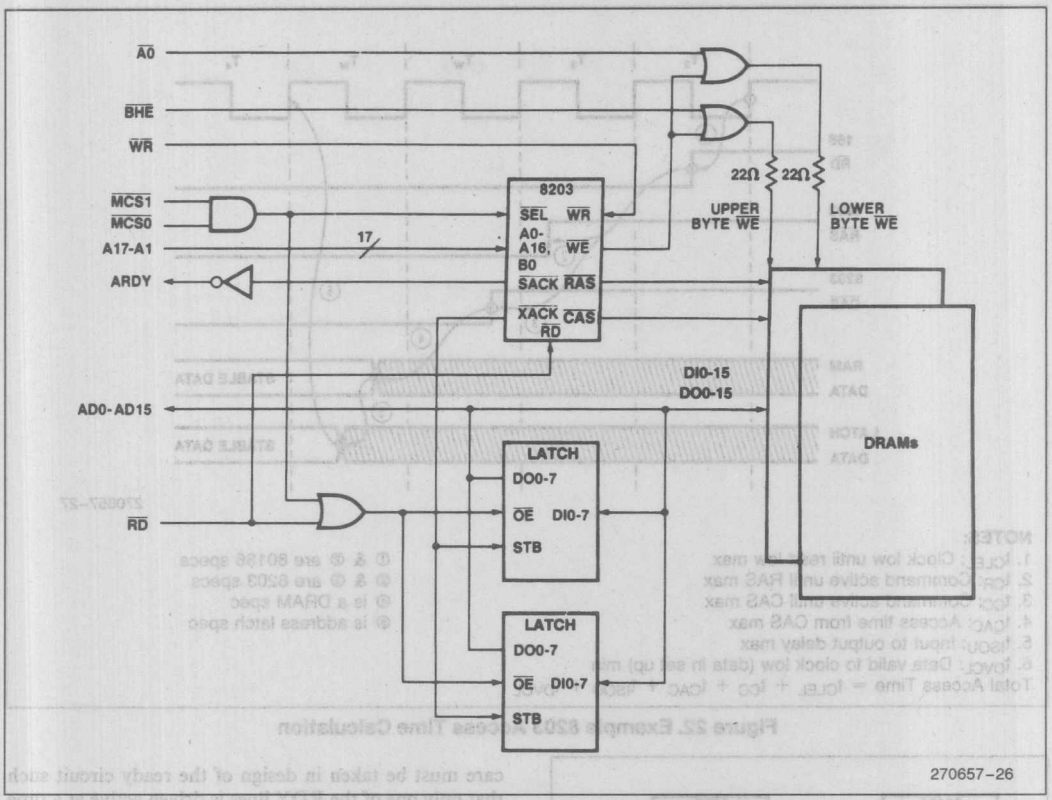


Figure 21. Example 8203/DRAM/80186 Interface

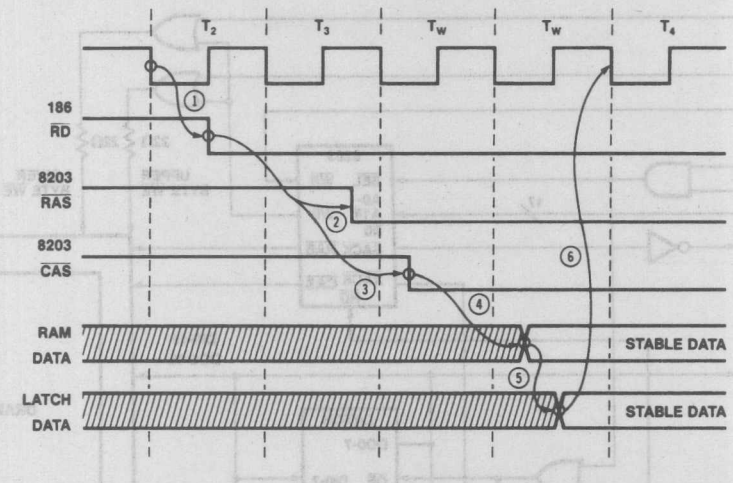
Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 must be synchronized to the 80186. The 80186 ARDY line provides the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. The 8203 $\overline{\text{SACK}}$ is presented to the 80186 only when the DRAM is being accessed. Notice that the $\overline{\text{SACK}}$ output of the 8203 is used, rather than $\overline{\text{XACK}}$. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active and the time data must be present on the data bus, the $\overline{\text{XACK}}$ signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

3.2.3 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 microcomputer systems. This controller provides all address multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 23. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array driven by the 8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

The 8207 $\overline{\text{AACK}}$ signal generates a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 ac-



NOTES:
1. t_{CLEL}: Clock low until read low max
2. t_{CR}: Command active until RAS max
3. t_{CC}: Command active until CAS max
4. t_{CAC}: Access time from CAS max
5. t_{ISOU}: Input to output delay max
6. t_{DVCL}: Data valid to clock low (data in set up) min

Total Access Time = $t_{CLEL} + t_{CC} + t_{CAC} + t_{ISOU} + t_{DVCL}$

① & ② are 80186 specs
② & ③ are 8203 specs
④ is a DRAM spec
⑤ is address latch spec

Figure 22. Example 8203 Access Time Calculation

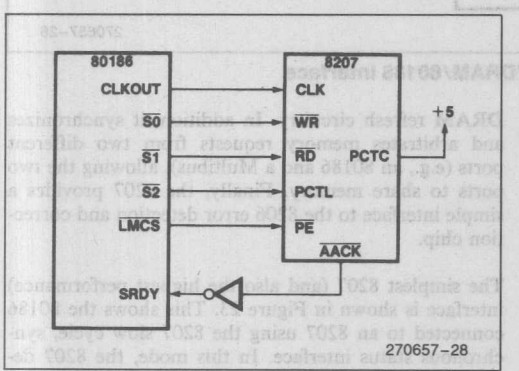


Figure 23. 80186/8207/DRAM Interface

cess cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the $\overline{AACK}$ line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that either SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus masters (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus.

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 24). AD0–AD15 and $\overline{DEN}$ are floated within

t_{CLAZ} after the same clock edge that HLDA is driven active. A16–A19, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, DT/$\overline{R}$, and $\overline{S0}$–$\overline{S2}$ are floated within t_{CHCZ} after the clock edge immediately before the clock edge on which HLDA comes active.

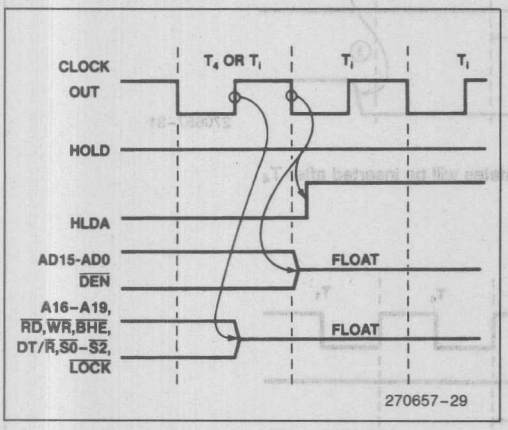


Figure 24. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TMR OUT). Many others either directly or indirectly control bus devices. These signals are ALE and all the chip select lines ($\overline{UCS}$, $\overline{LCS}$, MCS0–3, and $\overline{PCS0}$–6).

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect bus latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous input. To guarantee recognition on a particular clock edge, it must satisfy setup and hold times to the falling edge of the CPU clock. A full CPU clock cycle is required for synchronization (see Appendix B). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input;

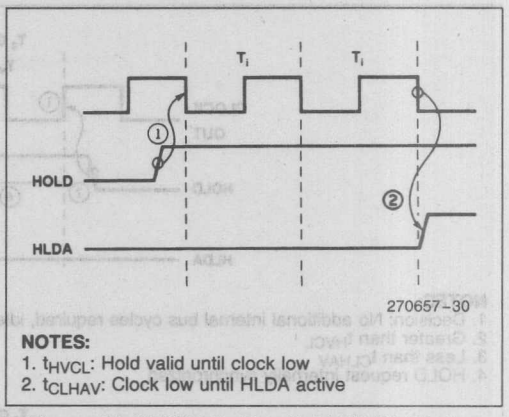
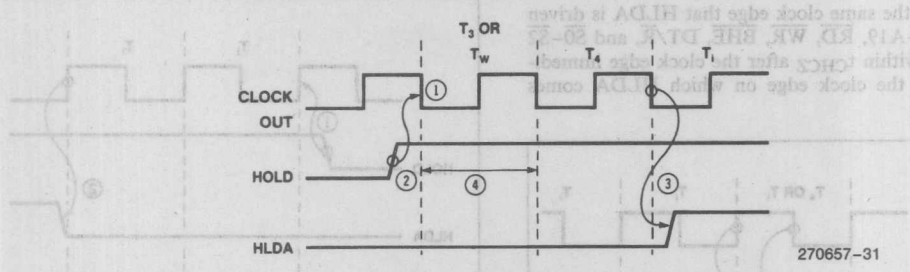


Figure 25. 80186 Idle Bus Hold/HLDA Timing

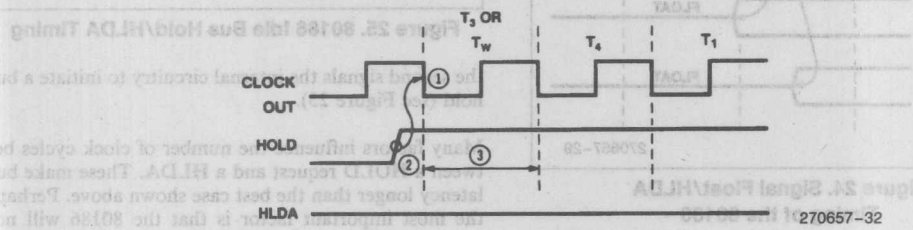
the second signals the internal circuitry to initiate a bus hold (see Figure 25).

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. The bus can become idle only at the end of a bus cycle. The 80186 will normally insert no T_i states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_i states independent of the HOLD input (see Section 3.1.1).

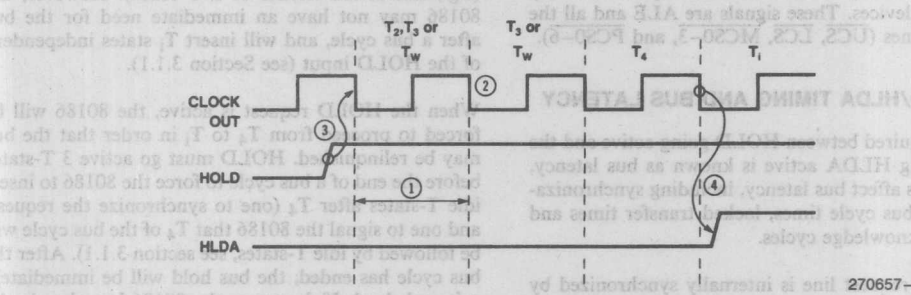
When the HOLD request is active, the 80186 will be forced to proceed from T_4 to T_i in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 26 graphically portrays the scenarios depicted above.



NOTES:
1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than t_{HVCL}
3. Less than t_{CLHAV}
4. HOLD request internally synchronized



NOTES:
1. Decision: Additional internal bus cycles required, no idle T-states will be inserted, HOLD not active soon enough to force idle T-states
2. Greater than t_{HVCL}: not required since it will not get recognized anyway
3. HOLD request internally synchronized



NOTES:
1. HOLD request internally synchronized
2. Decision: HOLD request active, idle t-states will be inserted at end of current bus cycle
3. Greater than t_{HVCL}
4. Less than t_{CLHAV}

Figure 26. HOLD/HLDA Timing in the 80186

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see Section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required for the integrated DMA unit to perform a transfer. Each of these factors will add to the bus latency of the 80186.

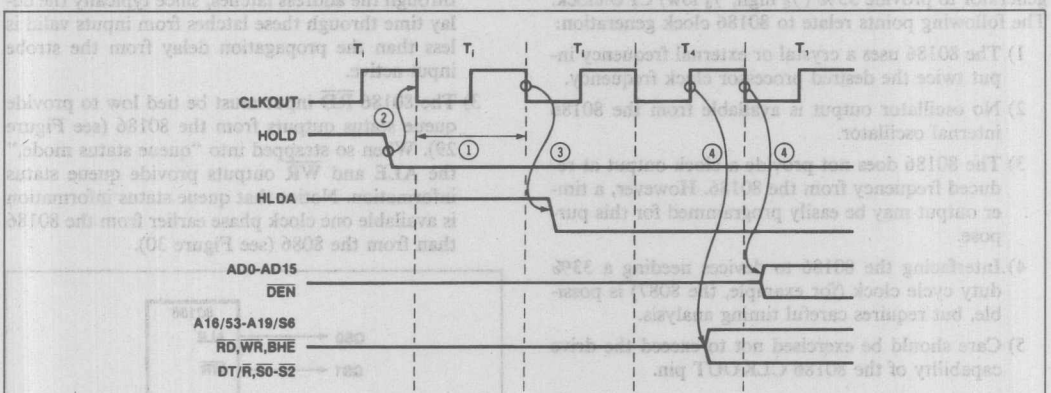
Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in Slave mode (see Section 4.4.1) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by bus HOLD. See Section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

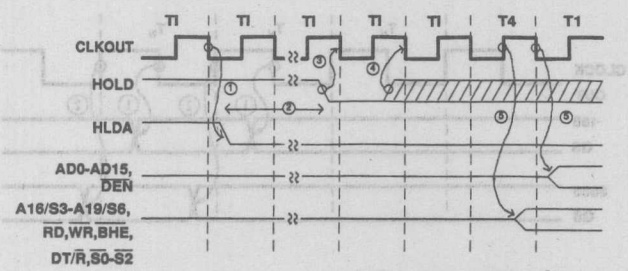
When the HOLD input goes inactive, the processor lowers its HLDA line in a single clock as shown in Figure 27. If there is pending bus activity, only two T_i states will be inserted after HLDA goes inactive and status information will go active during the last idle state concerning the bus cycle about to be run (see Section 3.1.1). If there are no bus cycles to be run by the CPU, it will continue to float all lines until the last T_i before it begins its first bus cycle after the HOLD.

A special mechanism exists on the 80C186/80C188 to provide for DRAM refreshing while the bus is in HOLD. If the refresh control unit issues a request to the integrated bus controller while HOLD is in effect, the processor lowers HLDA. It is the responsibility of the external bus master to release the bus by deasserting HOLD so that the refresh cycle can take place (see Figure 28). The external master can then reassume control of the bus subject to the usual requirements placed on the HOLD input.



NOTES:
1. HOLD internally synchronized
2. Greater than T_{HVCL}
3. Less than T_{CLHAV}
4. Lines come out of float only if a bus cycle is pending

Figure 27. 80186 Coming out of Hold



NOTES:
1. HLDA deasserted, signaling need to run DRAM refresh cycle; less than t_{CLHAV}.
2. External bus master terminates use of the bus.
3. HOLD deasserted; greater than t_{HVCL}.
4. HOLD may be reasserted after one clock.
5. Lines come out of float in order to run DRAM refresh cycle.

Figure 28. Release of 80C186/80C188 HOLD to Run Refresh Cycle

3.4 Differences between the 8086 Bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 82C88 bus controller and the 8289 bus arbiter) may be used with the 80186. There are a few significant differences between the two processors which should be considered.

CPU Duty Cycle and Clock Generator

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock. This is different from the 8086, which utilizes an external clock generator to provide 33% (1/3 high, 2/3 low) CPU clock. The following points relate to 80186 clock generation:

1) The 80186 uses a crystal or external frequency input twice the desired processor clock frequency.
2) No oscillator output is available from the 80186 internal oscillator.
3) The 80186 does not provide a clock output at reduced frequency from the 80186. However, a timer output may be easily programmed for this purpose.
4) Interfacing the 80186 to devices needing a 33% duty cycle clock (for example, the 8087) is possible, but requires careful timing analysis.
5) Care should be exercised not to exceed the drive capability of the 80186 CLKOUT pin.

Local Bus Controller and Control Signals

The 80186 simultaneously provides both local bus controller outputs and status outputs for use with the 82C88 bus controller. This is different from the 8086 where the local bus controller outputs are sacrificed if status outputs are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g., a MULTIBUS®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 82C88 bus controller to drive the control signals of the system bus.
2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 then on the 8086 or 82C88. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.
3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

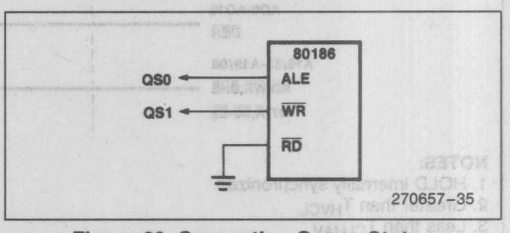
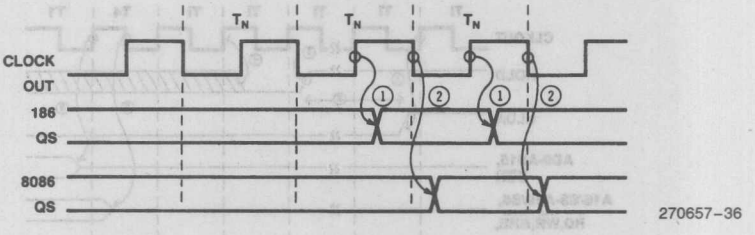


Figure 29. Generating Queue Status Information from the 80186



NOTES:
1. 80186 changes queue status off falling edge of CLK
2. 8086 changes queue status off rising edge of CLK

Figure 30. 80186 and 8086 Queue Status Generation

HOLD/HLDA vs. RQ/GT

As discussed earlier, the 80186 uses a HOLD/HLDA protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatibility with Intel's bus master peripheral devices (for example the 82586 Ethernet controller or 82730 high performance CRT controller/text coprocessor).

Status Information

The 80186 does not provide S3–S5 status information. On the 8086, S3 and S4 provide information regarding the segment register generating the physical address of the current bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status lines are always low on the 80186.

Status signal S6 indicates whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the integrated DMA unit.

Miscellaneous

The 80186 does not provide early and late write signals, as does the 82C88 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 82C88. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide both I/O and memory read and write command signals. If these signals are desired, an external 82C88 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize both commands (see Section 3.1.6.1).

4.0 DMA UNIT INTERFACING

The 80186 includes a DMA unit consisting of two independent DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are the same as bus cycles initiated by the CPU (except that S6 = 1 during all DMA initiated cycles). Interfacing the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

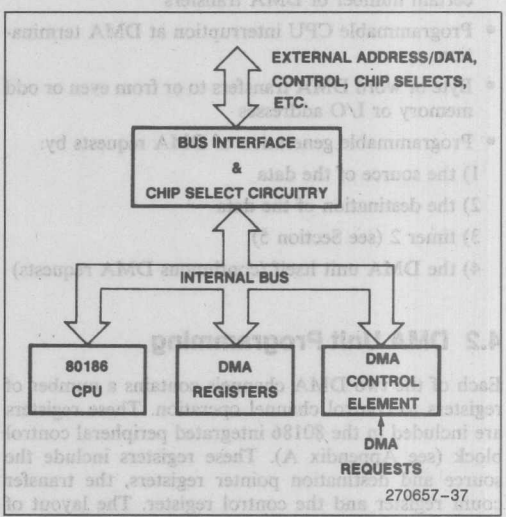


Figure 31. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
  1) the source of the data
  2) the destination of the data
  3) timer 2 (see Section 5)
  4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers to control channel operation. These registers are included in the 80186 integrated peripheral control block (see Appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout of the bits in these registers is given in Figures 32 and 33.

The 20-bit source and destination pointers access the complete 1 Mbyte address space of the 80186 and all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA. The address space is seen as a flat, linear array without segments. Even though the usual I/O addressability of the 80186 is 64 Kbytes, it is possible to perform I/O accesses over a 1 Mbyte address range. Therefore, it is important to program the upper four bits of the pointer registers to 0 if routine I/O addresses are desired.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/$\overline{\text{STOP}}$ bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

Upon reset, the contents of the DMA pointer registers and transfer count registers are indeterminate; initialization of all the bits should be practiced.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects byte or word transfers. Two synchronization bits determine the source of the DMA requests (see Section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit enables interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

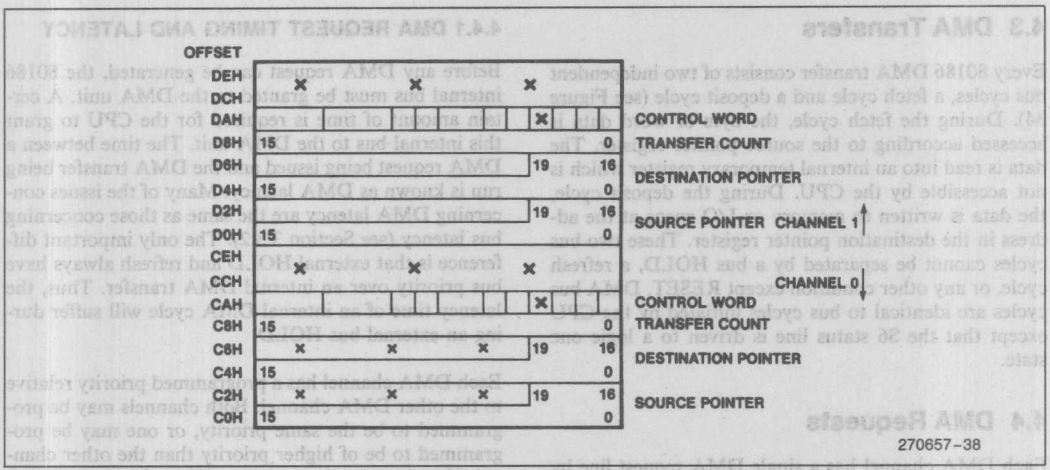


Figure 32. 80186 DMA Register Layout

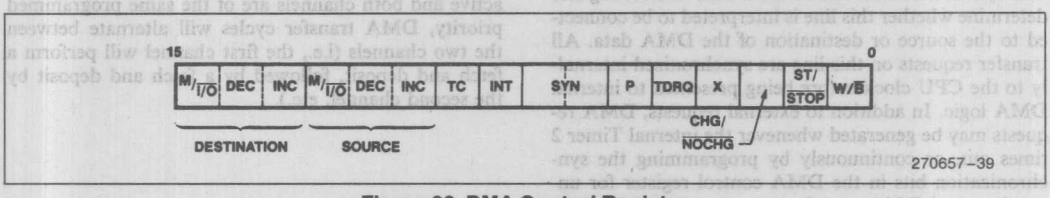


Figure 33. DMA Control Register

The control register also contains a start/stop (ST/$\overline{\text{STOP}}$) bit which enables DMA transfers. Whenever this bit is set, the channel is armed, that is, a DMA transfer will occur whenever a DMA request is made to the channel. A companion bit, the CHG/$\overline{\text{NOCHG}}$ bit, allows the DMA control register to be changed without modifying the state of the start/stop bit. The ST/$\overline{\text{STOP}}$ bit will only be modified if the CHG/$\overline{\text{NOCHG}}$ bit is also set during the write to the DMA control register. The CHG/$\overline{\text{NOCHG}}$ bit is write only. It will always be read back as a 0. Because DMA transfers could occur immediately after the ST/$\overline{\text{STOP}}$ bit is set, it should only be set after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessible by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/$\overline{\text{STOP}}$ bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only some of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).

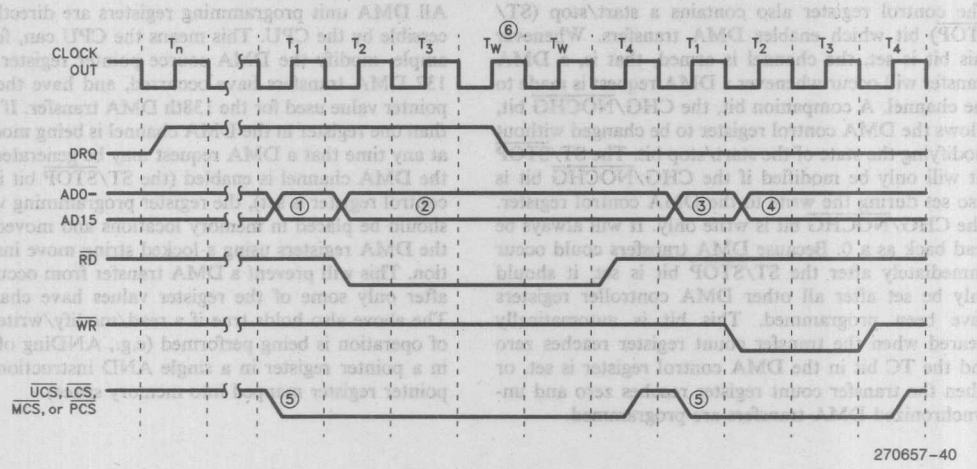
Every 80186 DMA transfer consists of two independent bus cycles, a fetch cycle and a deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed according to the source pointer register. The data is read into an internal temporary register which is not accessible by the CPU. During the deposit cycle, the data is written to memory or I/O space at the address in the destination pointer register. These two bus cycles cannot be separated by a bus HOLD, a refresh cycle, or any other condition except RESET. DMA bus cycles are identical to bus cycles initiated by the CPU except that the S6 status line is driven to a logic one state.

4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source or destination of the DMA data. All transfer requests on this line are synchronized internally to the CPU clock before being presented to internal DMA logic. In addition to external requests, DMA requests may be generated whenever the internal Timer 2 times out, or continuously by programming the synchronization bits in the DMA control register for unsynchronized DMA transfers.

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see Section 3.3.2). The only important difference is that external HOLD and refresh always have bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a fetch and deposit, followed by a fetch and deposit by the second channel, etc.).



NOTES:
1. Source address
2. Source data
3. Destination address
4. Destination data
5. If a source or destination address overlaps an active chip select region, the chip select will go active.
6. Wait states are inserted by programming the chip select/ready logic for an active address region, or by the external ready pins.

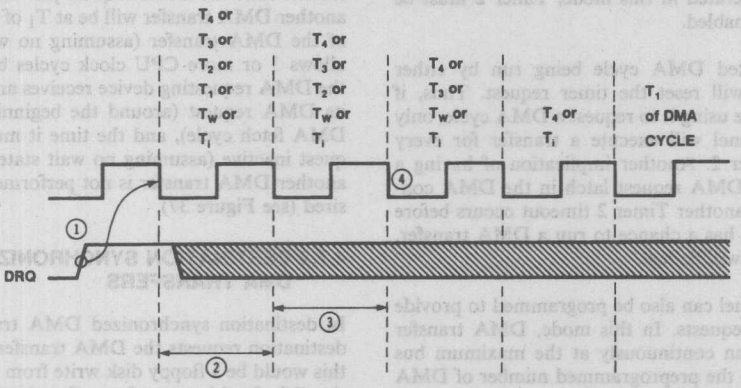
Figure 34. Example DMA Transfer Cycle on the 80186

The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity.

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor. Quite the contrary, DRQ must be active at a certain time before the end of a bus cycle for the request to be recognized by the processor. If DRQ goes inactive before that window, then no DMA cycles will be run.

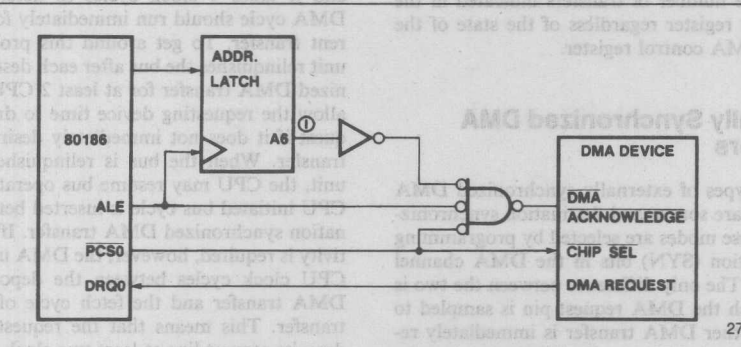
4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge (DACK) signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the $\overline{PCS}$ lines (see Figure 36). Note ALE must be used to factor DACK because addresses are not guaranteed stable when chip selects go active.



NOTES:
1. t_{INVCL} = DMA request to clock low
2. Synchronizer resolution time
3. DMA unit priority arbitration, etc. time
4. Bus Interface Unit latches DMA request and decides to run DMA cycle

Figure 35. DMA Request Timing on the 80186 (showing minimum response time to request)



NOTE:
1. A6 is arbitrary and is only used to differentiate DMA and non-DMA bus cycles

Figure 36. DMA Acknowledge Synthesis from the 80186

4.6 Internally Generated DMA Requests

DMA transfer requests may originate from two of the integrated peripherals in the 80186. The source may be either the DMA control unit or Timer 2.

The DMA channel can be programmed so that whenever Timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in **both** DMA control registers. Before any DMA requests are generated in this mode, Timer 2 must be initialized and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of Timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another Timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, until the preprogrammed number of DMA transfers have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

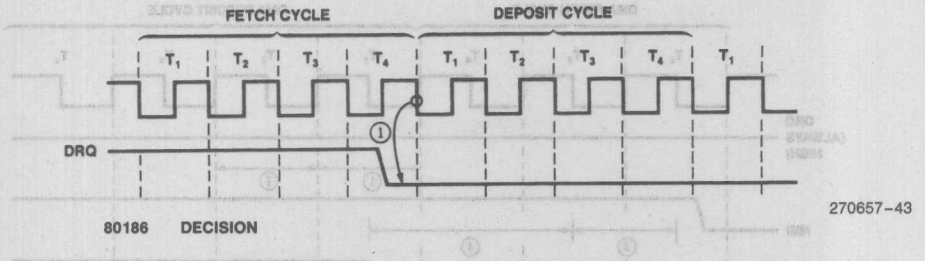
There are two types of externally synchronized DMA transfers. These are source and destination synchronized transfers. These modes are selected by programming the synchronization (SYN) bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

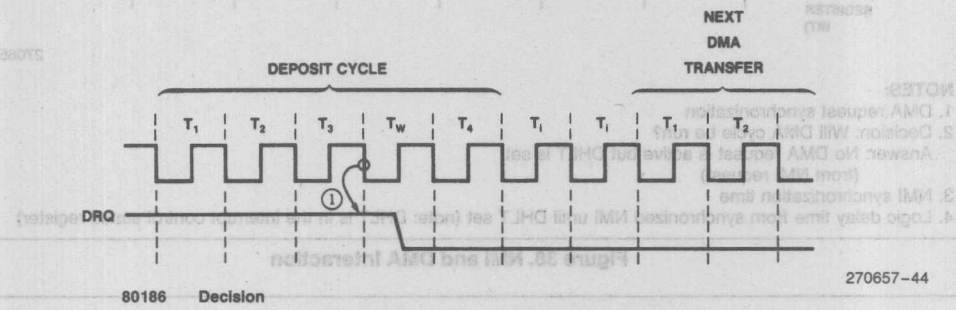
In a source synchronized DMA transfer, the data source requests the DMA cycle. An example is a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at T_1 of the deposit cycle of the DMA transfer (assuming no wait states.) This allows 3 or more CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the data destination requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the deposit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine if another DMA cycle should run immediately following the current transfer. To get around this problem, the DMA unit relinquishes the bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the requesting device time to drop its DMA request if it does not immediately desire another DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation. Typically, a CPU initiated bus cycle is inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however, the DMA unit inserts only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the requesting device must drop its request line at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted. Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted



NOTE:
1. Current DMA source synchronized transfer will not be immediately followed by another DMA transfer



NOTE:
1. Current DMA destination synchronized transfer will be followed immediately by another DMA transfer

Figure 37. Source & Destination Synchronized DMA Request Timing

in the deposit cycle of the transfer lengthen the amount of time from the beginning of the deposit cycle to the time DRQ is sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its request after receiving an acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (about 1 clock), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its request after receiving the DMA acknowledge.

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended at the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see Section 6.3.7). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit can be cleared by the programmer to resume DMA activity (i.e., it is not automatically cleared when entering the NMI service routine). The DHLT bit is automatically cleared when the IRET instruction is executed. In either case, DMA activity resumes exactly as it left off, i.e., none of the DMA control registers are modified. This DHLT bit may also be set by the programmer to prevent DMA activity during critical sections of code. The DHLT bit does not function when the integrated interrupt controller is configured for Slave Mode.

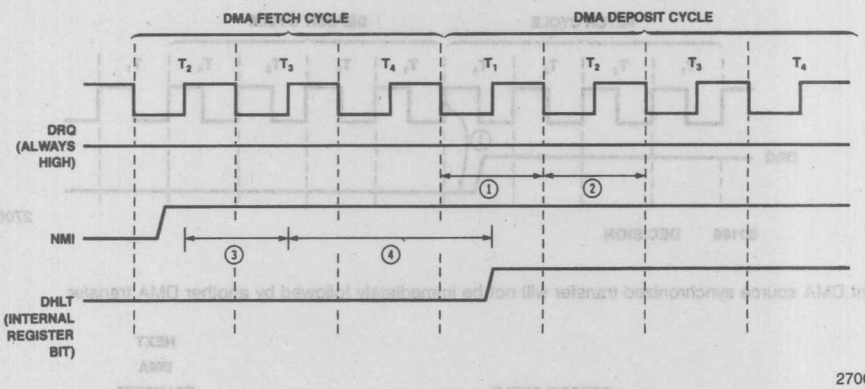
4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0–AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 upon the next bus cycle (see Section 3.1.5).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to $\overline{PCS2}$. If $\overline{PCS2}$ is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0501H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored



NOTES:
1. DMA request synchronization
2. Decision: Will DMA cycle be run?
   Answer: No DMA request is active but DHLT is set
   (from NMI request)
3. NMI synchronization time
4. Logic delay time from synchronized NMI until DHLT set (note: DHLT is in the interrupt control status register)

Figure 38. NMI and DMA Interaction

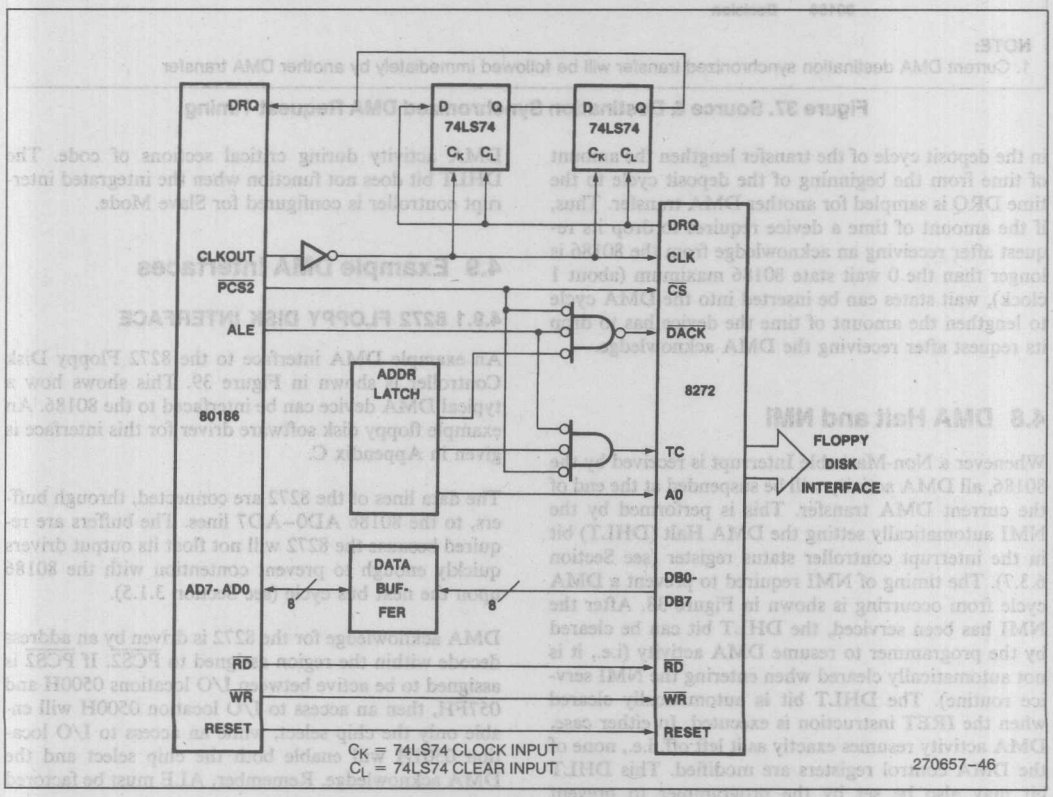


Figure 39. Example 8272/80186 DMA Interface

into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to output a pulse to the 8272 after a certain number of DMA cycles have been run (see next section for 80186 timer information).

The above discussion assumed that a single 80186 $\overline{PCS}$ line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated $\overline{PCS}$ lines could be used for each function. For example, $\overline{PCS2}$ could be used to select the 8272 and $\overline{PCS3}$ could be used to drive the DACK line of the 8272.

DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{RD}$ going active) spec. This requires many 80186 CPU clock cycles, well beyond the 5 minimum provided by the 80186 (4 clock cycles to the beginning of the DMA bus cycle, 5 to the beginning of T_2 of the DMA bus cycle where $\overline{RD}$ will go active). The two flip-flops add two complete CPU clock cycles to this response time.

DMA request will go away after DACK is presented to the 8272. During a DMA write cycle (i.e., a destination synchronized transfer), this does not occur soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 40. The 8274 interface is simpler than the 8272 interface, since it does not require a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.

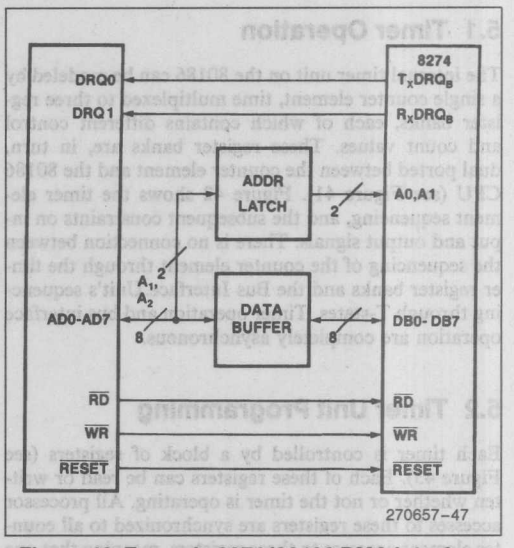


Figure 40. Example 8274/80186 DMA Interface

The data lines of the 8274 are connected through buffers to the 80186 AD0–AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 does not float its drivers before the 80186 begins driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read or write of the 8274 data register after the 8274 generates the DMA request signal clears the DMA request. The time between the control signal ($\overline{RD}$ or $\overline{WR}$) going active and the 8274 dropping its DMA request during a DMA write requires at least one wait state be inserted into the DMA write cycle.

5.0 TIMER UNIT INTERFACING

The 80186 includes a timer unit which consists of three independent 16-bit timers. These timers operate independently of the CPU. Two have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 can be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Unit Programming

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter elements from accessing the timer registers.

Each timer has a 16-bit count register which is incremented for each timer event. A timer event can be a low-to-high transition on a TIMERIN pin (for Timers 0 and 1), a pulse generated every fourth CPU clock, or a time out of Timer 2 (for Timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted. Upon RESET, the contents of the count registers are indeterminate and they should be initialized to zero before any timer operation.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register resets to zero, so the maximum count value can never be stored in the count register. This maximum count value may be written while the timer is operating. A maximum count value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. Only equivalence between the count value and the maximum count register value is checked. This means that the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This situation only occurs by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If the timer is programmed in this way, it will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

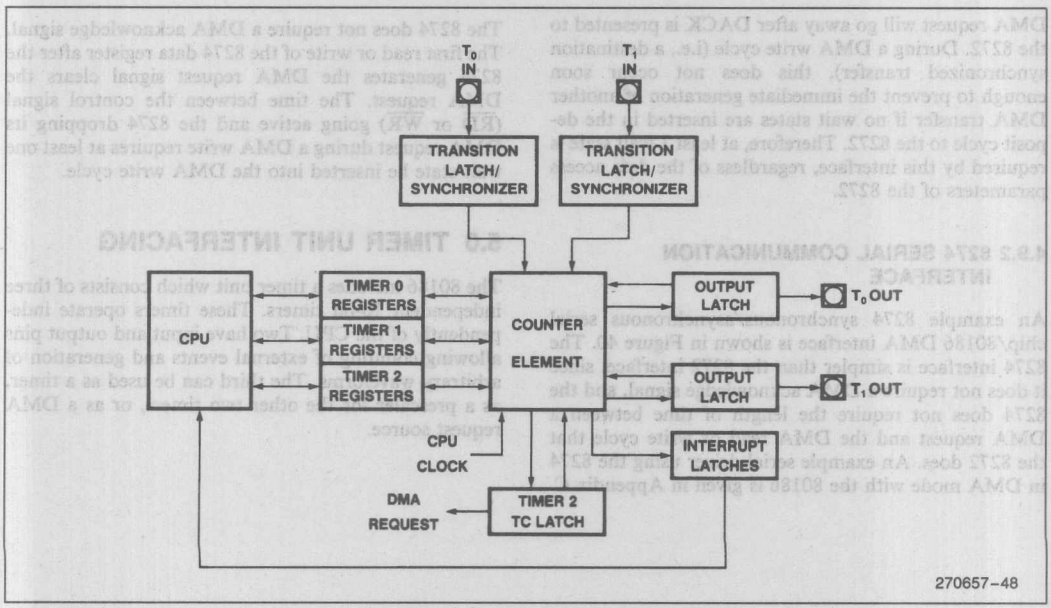
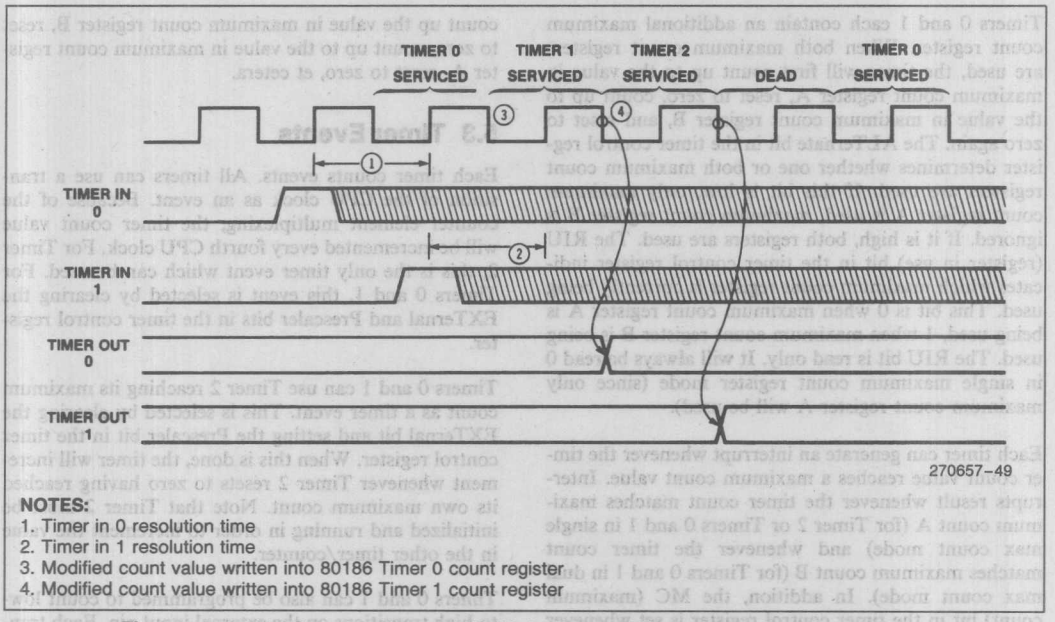


Figure 41. 80186 Timer Model



NOTES:
1. Timer in 0 resolution time
2. Timer in 1 resolution time
3. Modified count value written into 80186 Timer 0 count register
4. Modified count value written into 80186 Timer 1 count register

Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

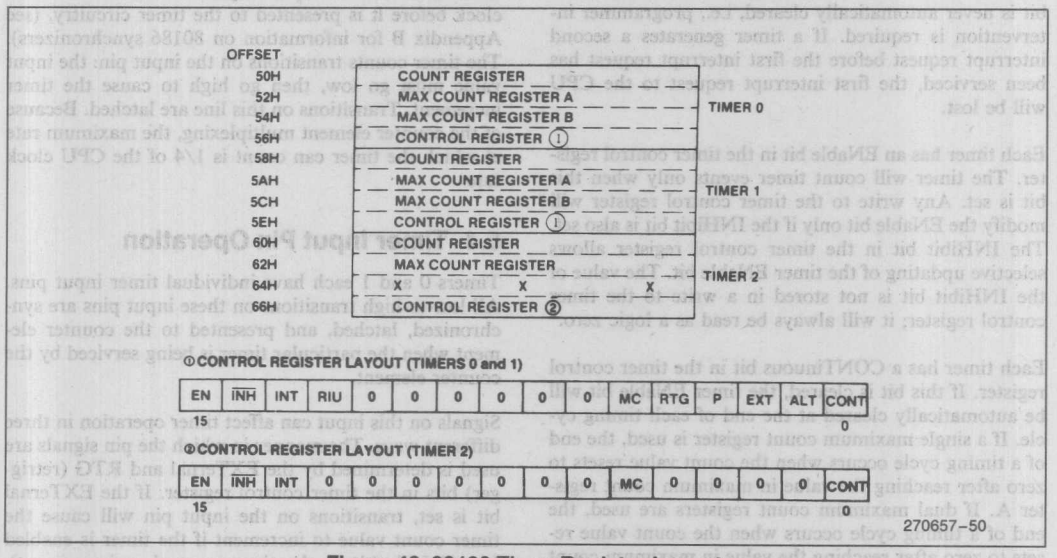


Figure 43. 80186 Timer Register Layout

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTernate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both registers are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. The RIU bit is read only. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. Interrupts result whenever the timer count matches maximum count A (for Timer 2 or Timers 0 and 1 in single max count mode) and whenever the timer count matches maximum count B (for Timers 0 and 1 in dual max count mode). In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. The timer will count timer events only when this bit is set. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a logic zero.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up the value in maximum count register A, reset to zero, count up the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For Timer 2, this is the only timer event which can be used. For Timers 0 and 1, this event is selected by clearing the EXTernal and Prescaler bits in the timer control register.

Timers 0 and 1 can use Timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTernal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever Timer 2 resets to zero having reached its own maximum count. Note that Timer 2 must be initialized and running in order to increment the value in the other timer/counter.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer increment. Transitions on this line are latched. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate.

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTernal and RTG (retrigger) bits in the timer control register. If the EXTernal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTernal bit is cleared, all timer increments are caused by either the CPU clock or by Timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

When the EXTernal bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: **both** must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

When the EXTernal bit is low and the RTG bit is high, every low-to-high transition on the timer input pin causes the timer count register to reset to zero. This mode of operation can be used to generate a retriggerable digital one-shot. After the timer is enabled (i.e., the ENable bit in the timer control register is set), timer operation (counting) will begin **only after** the first low-to-high transition of the timer input pin has been detected. If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timer cycle again. A timer cycle is defined as the time the timer is counting from 0 to the maximum count (either max count A or max count B). This means that in the dual max count mode, the RIU bit is not set if the timer is reset by the low-to-high transition on the input pin. Should a timer reset occur when RIU is set (indicating max count B), the timer will again begin to count up to max count B before resetting the RIU bit. Thus, when the ALTernate bit is set, a timer reset will retrigger (or extend) the duration of the current max count in use (which means that either the low or high level of the timer output will be extended). If the CONTinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared whenever a timer cycle has been completed (max count is reached). If the CONTinuous bit in the timer control register is set, the timer will reset to zero and begin another timer cycle whenever the current cycle has completed.

5.5 Timer Output Pin Operation

Timers 0 and 1 each have a timer output pin which can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer being serviced by the counter element.

When the timer is in single maximum count register mode, the timer output pin will go low for a single CPU clock one clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode, the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used and high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTinuous bit in the timer control register is set), this pin could generate a waveform of almost any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

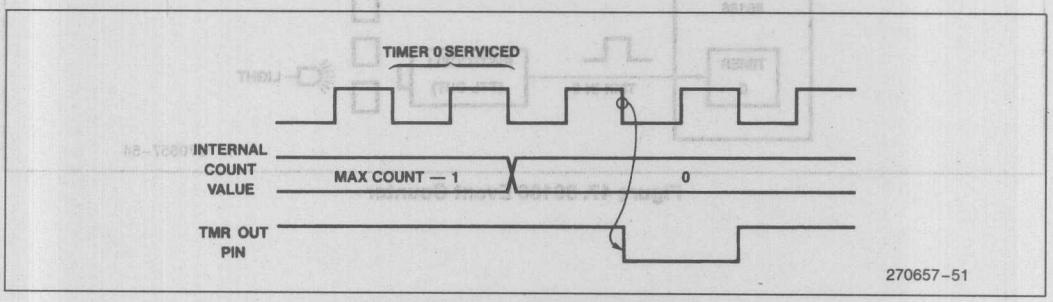


Figure 44. 80186 Timer Out Signal

5.6 Sample 80186 Timer Applications

The 80186 timers can be substituted in almost any application for a discrete timer circuit. Such applications include baud rate generation, digital one-shots, pulse width modulation, event counters and pulse width measurement.

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, Timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

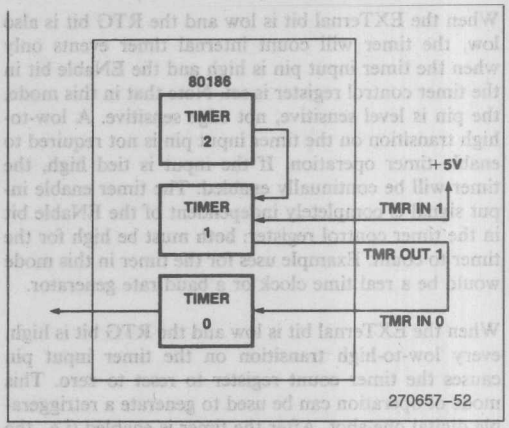


Figure 45. 80186 Real Time Clock

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the code to program the timer as a baud rate generator is included in Appendix D.

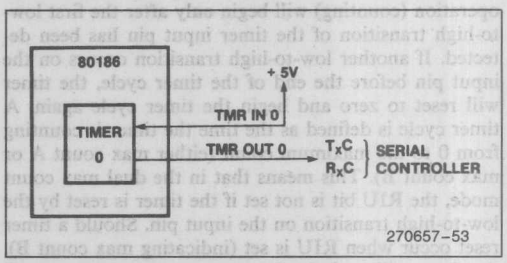


Figure 46. 80186 Baud Rate Generator

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in Appendix D.

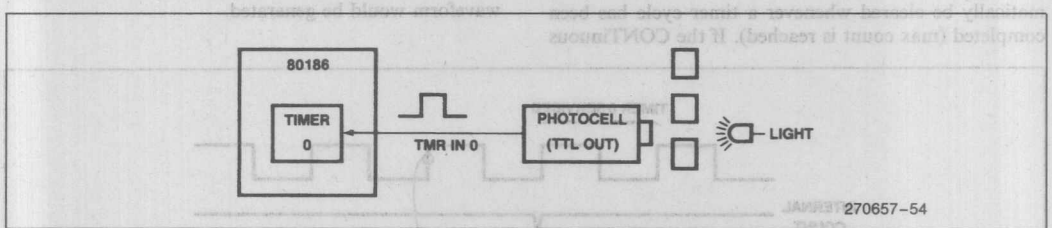


Figure 47. 80186 Event Counter

6.0 80186 INTERRUPT CONTROLLER INTERFACING

The tasks performed by the 80186 integrated interrupt controller include synchronization of interrupt requests, prioritization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external master interrupt controller.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, Master Mode and Slave Mode. In Master Mode the integrated controller acts as the master interrupt controller for the system, while in Slave Mode the controller operates as a slave to an external master interrupt controller. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes. The difference is when in Master Mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in Slave Mode the interrupt controller presents an interrupt output to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in Slave Mode is done by setting the SLAVE/$\overline{\text{MASTER}}$ bit in the peripheral control block pointer (see Appendix A).

6.3 Interrupt Controller Unit Programming

The interrupt controller has a number of registers which control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller. The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two modes of operation. The interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers

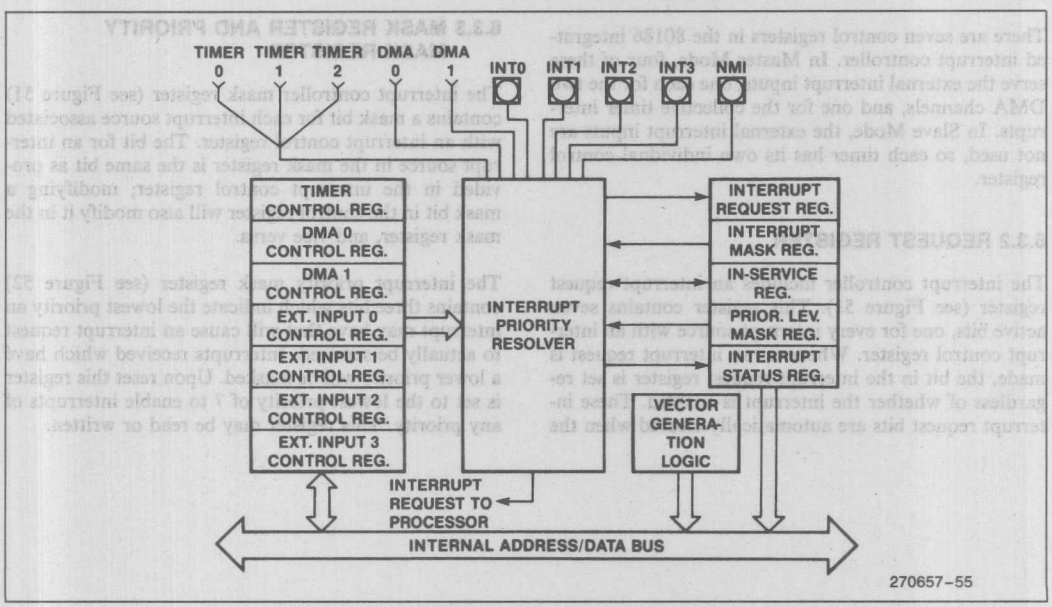
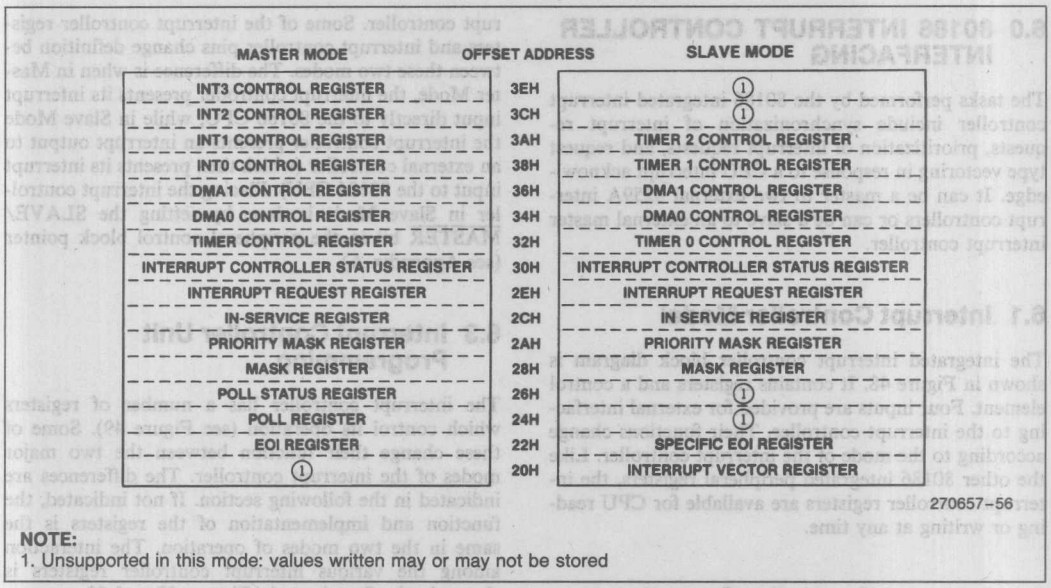


Figure 48. 80186 Interrupt Controller Block Diagram



NOTE:
1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers

contain three bits which select one of eight interrupt priority levels for the device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is zero, the interrupt is enabled, when it is one, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In Master Mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In Slave Mode, the external interrupt inputs are not used, so each timer has its own individual control register.

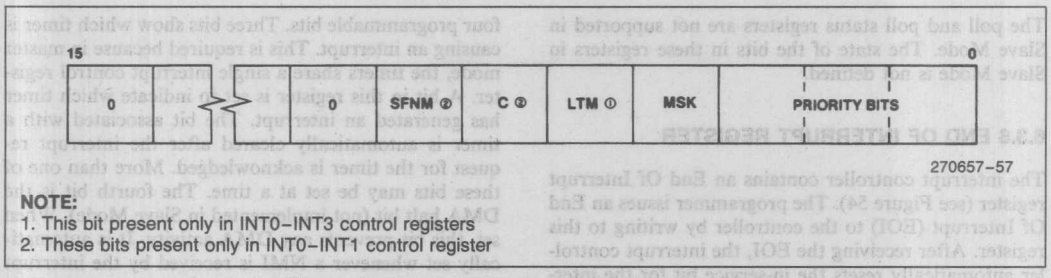
6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for every interrupt source with an interrupt control register. Whenever an interrupt request is made, the bit in the interrupt request register is set regardless of whether the interrupt is enabled. These interrupt request bits are automatically cleared when the interrupt is acknowledged. The D1 and D0 bits of the request register can also be set (requesting a DMA interrupt), or cleared (removing a DMA interrupt request) by programming.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller mask register (see Figure 51) contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is the same bit as provided in the interrupt control register; modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt priority mask register (see Figure 52) contains three bits which indicate the lowest priority an interrupt may have that will cause an interrupt request to actually be serviced. Interrupts received which have a lower priority will be masked. Upon reset this register is set to the lowest priority of 7 to enable interrupts of any priority. This register may be read or written.



NOTE:
1. This bit present only in INT0–INT3 control registers
2. These bits present only in INT0–INT1 control register

Figure 50. Interrupt Controller Control Register

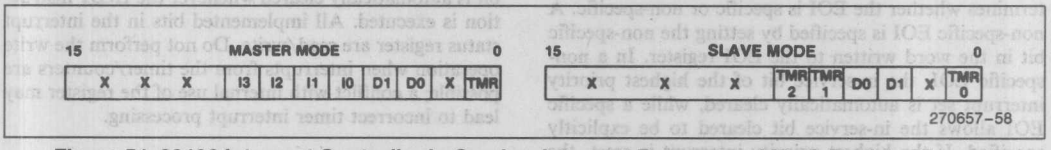


Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

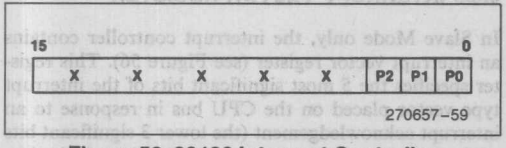


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the control register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). These registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If an interrupt is pending, the remaining bits contain information about the highest priority pending interrupt. These registers are read-only.

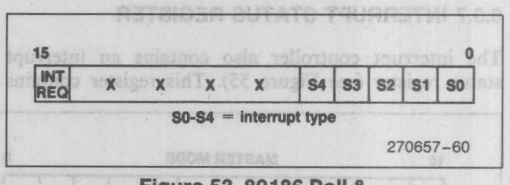


Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. The contents of the interrupt request, in-service, poll, and poll status registers will change appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers.

The poll and poll status registers are not supported in Slave Mode. The state of the bits in these registers in Slave Mode is not defined.

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt (EOI) to the controller by writing to this register. After receiving the EOI, the interrupt controller automatically resets the in-service bit for the interrupt. The value of the word written to this register determines whether the EOI is specific or non-specific. A non-specific EOI is specified by setting the non-specific bit in the word written to the EOI register. In a non-specific EOI, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific EOI allows the in-service bit cleared to be explicitly specified. If the highest priority interrupt is reset, the poll and poll status registers change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority poll and poll status registers will not be modified (because the highest priority interrupt to be serviced has not changed). Only the specific EOI is supported in Slave Mode. This register is write only.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains four programmable bits. Three bits show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit is the DMA halt bit (not implemented in Slave Mode). When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All implemented bits in the interrupt status register are read/write. Do not perform the write operation when interrupts from the timer/counters are possible; a conflict with internal use of the register may lead to incorrect timer interrupt processing.

6.3.8 INTERRUPT VECTOR REGISTER

In Slave Mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register specifies the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in Slave Mode).

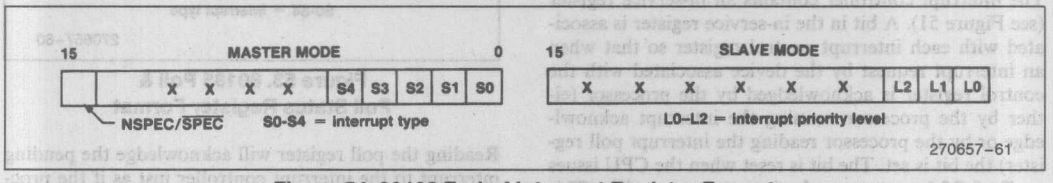


Figure 54. 80186 End of Interrupt Register Format

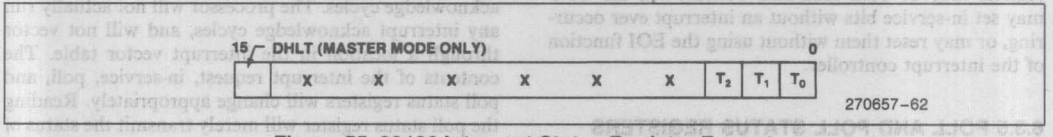


Figure 55. 80186 Interrupt Status Register Format

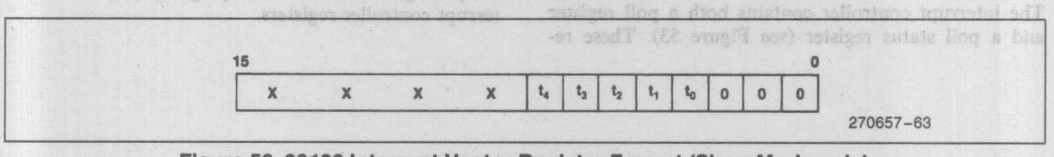


Figure 56. 80186 Interrupt Vector Register Format (Slave Mode only)

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Internal interrupts are processed by the interrupt controller in either Master Mode or Slave Mode. External interrupts are processed by the integrated interrupt controller only in Master Mode. Each interrupt source may be programmed to be a different priority level. An interrupt request generation flow chart is shown in Figure 57. This flowchart is followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register. Also, latched DMA interrupts can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request. Each timer has a unique interrupt vector (see Section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level relative to other interrupt sources. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lower priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in Master Mode. In this mode, the external pins associated

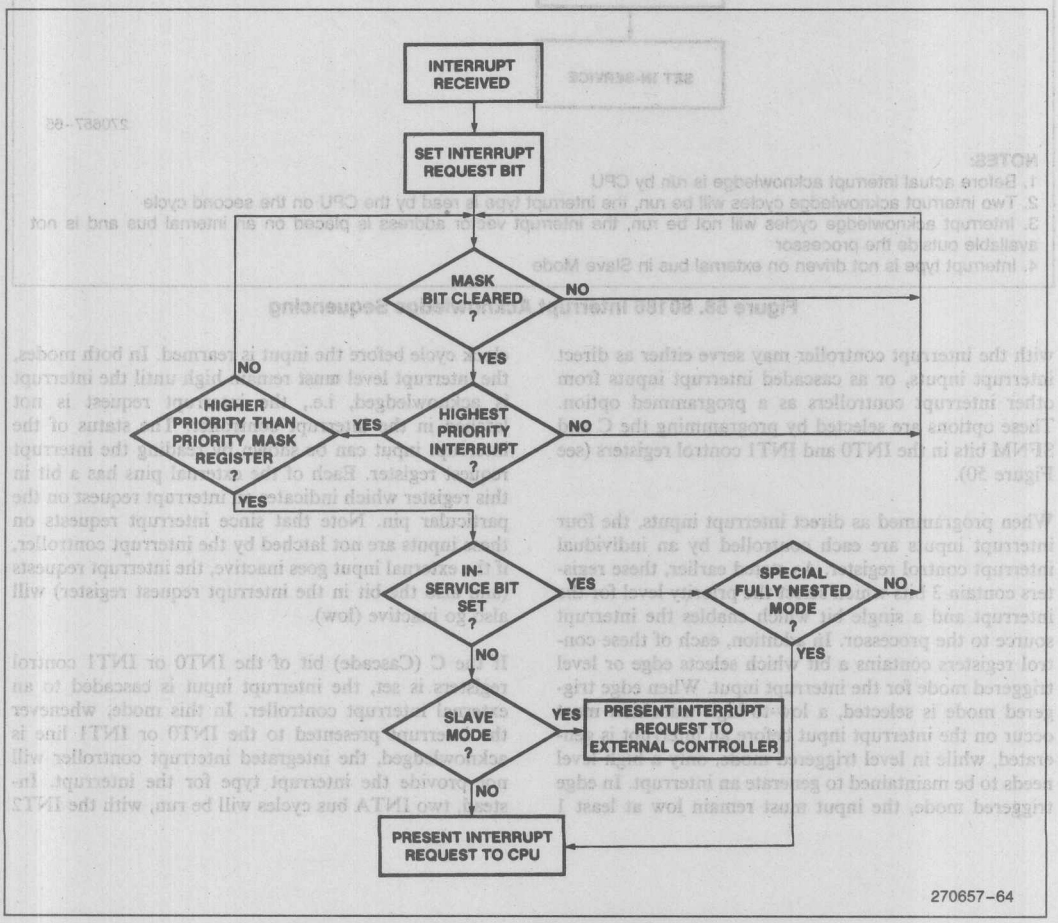
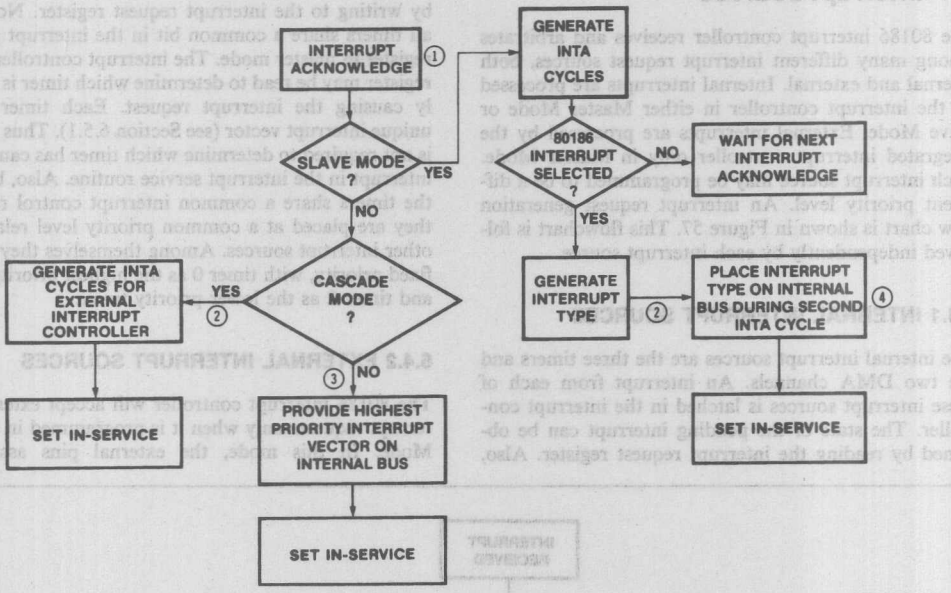


Figure 57. 80186 Interrupt Request Sequencing



NOTES:
1. Before actual interrupt acknowledge is run by CPU
2. Two interrupt acknowledge cycles will be run, the interrupt type is read by the CPU on the second cycle
3. Interrupt acknowledge cycles will not be run, the interrupt vector address is placed on an internal bus and is not available outside the processor
4. Interrupt type is not driven on external bus in Slave Mode

Figure 58. 80186 Interrupt Acknowledge Sequencing

with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition, each of these control registers contains a bit which selects edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition **must** occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is rearmed. In both modes, the interrupt level **must** remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt requests (and also the bit in the interrupt request register) will also go inactive (low).

If the C (Cascade) bit of the INT0 or INT1 control registers is set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2

and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into Cascade Mode. This allows 128 individually vectored interrupt sources if two banks of 8 external interrupt controllers each are used.

6.4.3 SLAVE MODE INTERRUPT SOURCES

When the interrupt controller is configured in Slave Mode, it accepts interrupt requests only from the integrated peripherals. Any external interrupt requests go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line. In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in Slave Mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see Section 6.6.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is providing the interrupt vector information with the controller in Master Mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in Slave Mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and reset the interrupt request bit. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In Master Mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4.

In Master Mode, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in memory.

Table 4. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Relative Priority
Timer 0	8	0(a)
Timer 1	18	0(b)
Timer 2	19	0(c)
DMA 0	10	1
DMA 1	11	2
INT 0	12	3
INT 1	13	4
INT 2	14	5
INT 3	15	6

Because two interrupt acknowledge cycles are not run, interrupt response to an internally vectored interrupt is 42 clock cycles. This is faster than the interrupt response when external vectoring is required, or if the interrupt controller is run in Slave Mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in Table 4) is used.

6.5.2 INTERNAL VECTORING, SLAVE MODE

In Slave Mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In Slave Mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowl-

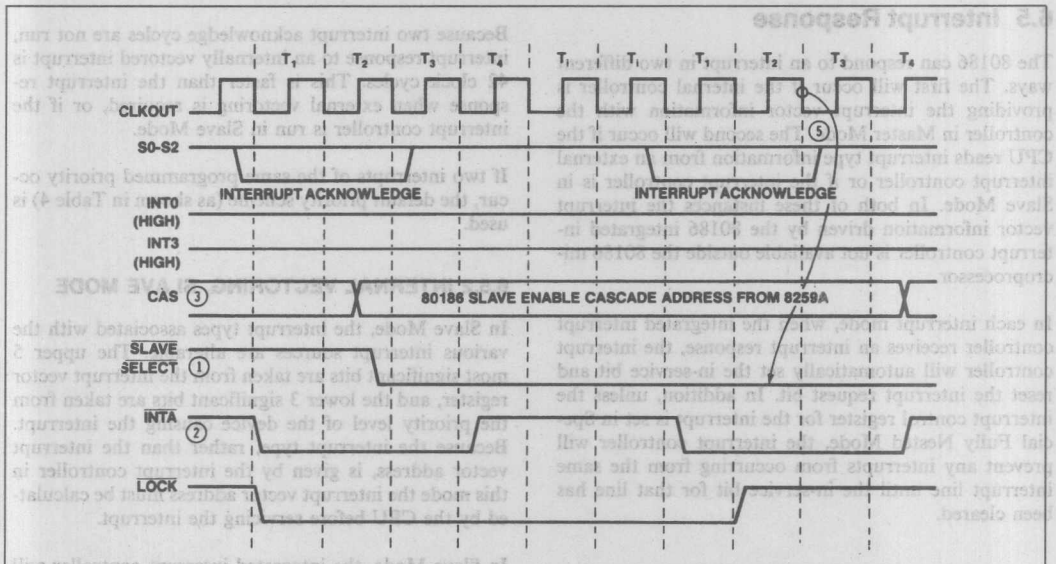
edge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186 samples the $\overline{\text{SLAVE SELECT}}$ line (INT1) during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable before and after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the $\overline{\text{LOCK}}$ output active. The two interrupt acknowledge cycles will always be separated by two idle T states, and wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in Slave Mode and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time is the same as for an externally vectored interrupt, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in Cascade Mode, Special Fully Nested Mode, or Slave Mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 59). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also



NOTES:
1. $\overline{\text{SLAVE SELECT}}$ = INT1
2. $\overline{\text{INTA}}$ = INT 2
3. Driven by external interrupt controller
4. $\overline{\text{SLAVE SELECT}}$ must be driven before Phase 2 of T_2 of the second INTA cycle
5. $\overline{\text{SLAVE SELECT}}$ read by 80186

Figure 59. 80186 Slave Mode Interrupt Acknowledge Timing

notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/$\overline{\text{INTA0}}$ and INT3/$\overline{\text{INTA1}}$ lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines ($\overline{\text{S0}}$–$\overline{\text{S2}}$) when either INT2/$\overline{\text{INTA0}}$ or INT3/$\overline{\text{INTA1}}$ signal an interrupt acknowledge.

6.5.4 EFFECT OF LOCK PREFIX ON INTERRUPT ACKNOWLEDGE CYCLES

When the interrupt controller is operating in either the cascade or slave modes and an interrupt occurs during an instruction that has been LOCKED by software, the LOCK signal timing shown in Figures 59 and 60 may be altered. Some peripheral devices used with the 80186 require contiguous $\overline{\text{INTA}}$ cycles to allow correct interrupt controller response. In such cases, the external circuitry in Figure 61 should be used to ensure that DMA or HOLD requests are blocked from stealing the bus during $\overline{\text{INTA}}$ cycles.

6.6 Interrupt Controller External Connections

The four interrupt signals can be configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or Slave Mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

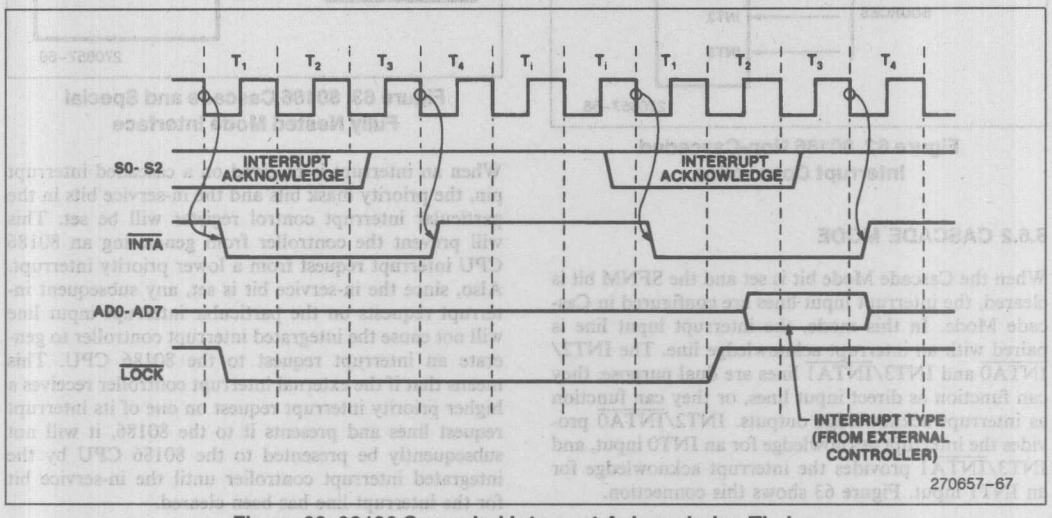


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

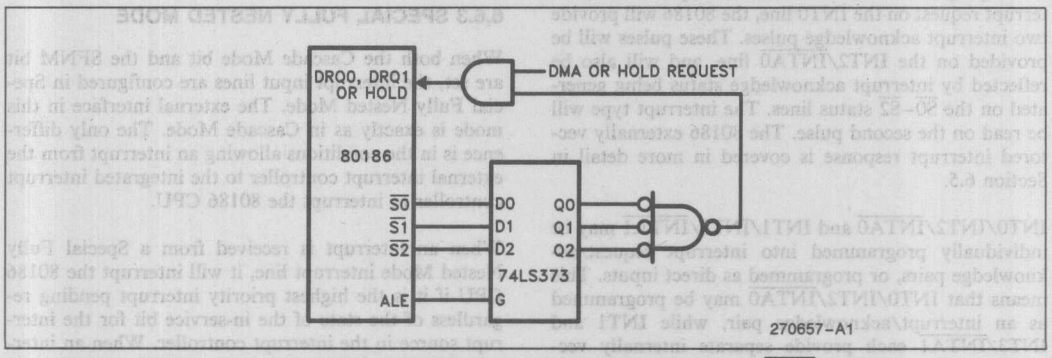


Figure 61. Circuit Blocking DMA or HOLD Request Between $\overline{\text{INTA}}$ Cycles

6.6.1 DIRECT INPUT MODE

When the Cascade Mode bits are cleared, the interrupt input pins are configured as direct interrupt pins (see Figure 62). Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any external interrupt acknowledge ($\overline{INTA}$) cycles.

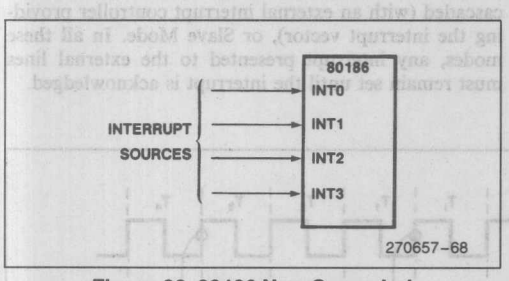


Figure 62. 80186 Non-Cascaded Interrupt Connection

6.6.2 CASCADE MODE

When the Cascade Mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in Cascade Mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/$\overline{INTA0}$ and INT3/$\overline{INTA1}$ lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/$\overline{INTA0}$ provides the interrupt acknowledge for an INT0 input, and INT3/$\overline{INTA1}$ provides the interrupt acknowledge for an INT1 input. Figure 63 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/$\overline{INTA0}$ line, and will also be reflected by interrupt acknowledge status being generated on the $\overline{S0}$–$\overline{S2}$ status lines. The interrupt type will be read on the second pulse. The 80186 externally vectored interrupt response is covered in more detail in Section 6.5.

INT0/INT2/$\overline{INTA0}$ and INT1/INT3/$\overline{INTA1}$ may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/$\overline{INTA0}$ may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/$\overline{INTA1}$ each provide separate internally vectored interrupt inputs.

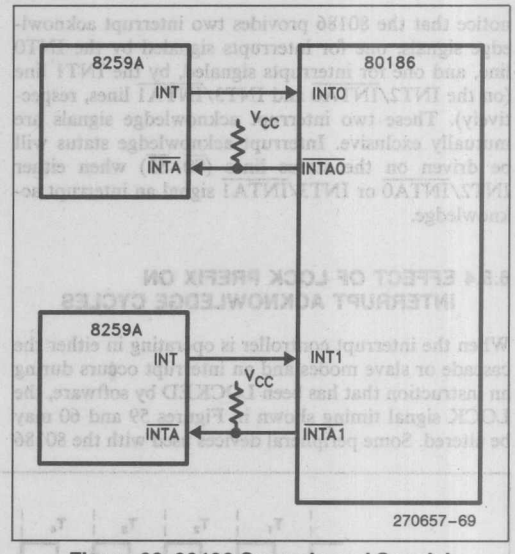


Figure 63. 80186 Cascade and Special Fully Nested Mode Interface

When an interrupt is received on a cascaded interrupt pin, the priority mask bits and the in-service bits in the particular interrupt control register will be set. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade Mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a Special Fully Nested Mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be

set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike Cascade Mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM bit is set and the Cascade Mode bit is not set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt. This operation is only applicable to INT0 and INT1, which have SFNM bits in their control registers.

6.6.4 SLAVE MODE

When the SLAVE/$\overline{\text{MASTER}}$ bit in the peripheral relocation register is set, the interrupt controller is in Slave Mode. In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 64 shows the hardware configuration of the 80186 interrupt lines with an external controller in Slave Mode.

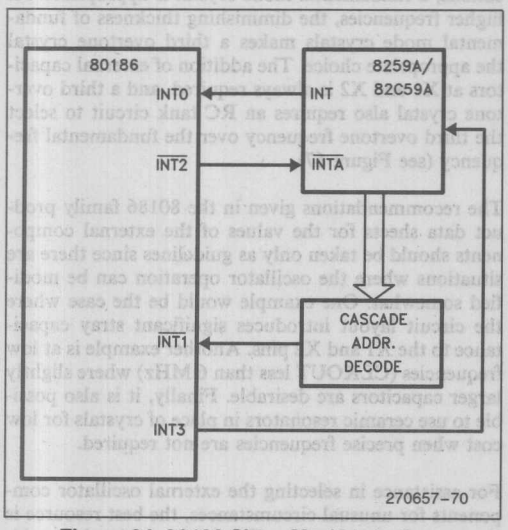


Figure 64. 80186 Slave Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/$\overline{\text{Slave Select}}$ lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/$\overline{\text{INTA0}}$ lines provide these two functions.

6.7 Example 8259A or 82C59A Cascade Mode Interface

Figure 65 shows the 80186 and 8259A (or 82C59A) in Cascade Mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an interrupt ready signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in Slave Mode).

6.8 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the intstruction being executed and the state of the interrupt enable flip-flop. The interrupt enable flip-flop must be explicitly set by issuing the STI instruction. Since interrupt vectoring automatically clears the flip-flop, it is necessary to set the flip-flop within the interrupt service routine if nested interrupts are desired.

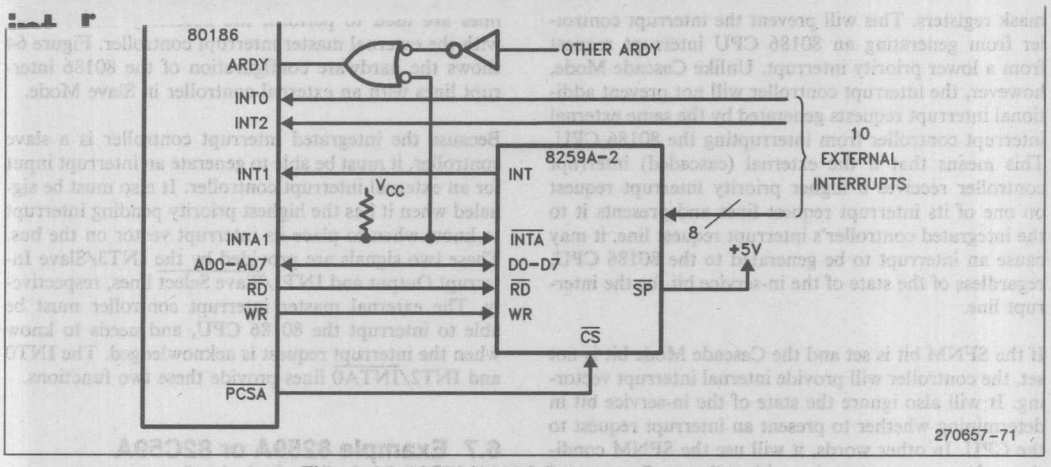


Figure 65. 80186/8259A Interrupt Cascading

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). An interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the $\overline{\text{TEST}}$ input is active. If the WAIT sees the $\overline{\text{TEST}}$ input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. Finally, the 80C186 and 80C188 will not accept interrupts during refresh bus cycles.

7.0 CLOCK GENERATOR

The 80186 clock generator provides the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

All 80186 family microprocessors use a parallel resonant Pierce oscillator. For all NMOS 80186/80188 applications and lower frequency 80C186/80C188 applications, a fundamental mode crystal is appropriate. At higher frequencies, the diminishing thickness of fundamental mode crystals makes a third overtone crystal the appropriate choice. The addition of external capacitors at X1 and X2 is always required, and a third overtone crystal also requires an RC tank circuit to select the third overtone frequency over the fundamental frequency (see Figure 67).

The recommendations given in the 80186 family product data sheets for the values of the external components should be taken only as guidelines since there are situations where the oscillator operation can be modified somewhat. One example would be the case where the circuit layout introduces significant stray capacitance to the X1 and X2 pins. Another example is at low frequencies (CLKOUT less than 6 MHz) where slightly larger capacitors are desirable. Finally, it is also possible to use ceramic resonators in place of crystals for low cost when precise frequencies are not required.

For assistance in selecting the external oscillator components for unusual circumstances, the best resource is the crystal manufacturer. The foremost circuit consideration is that the oscillator **start** correctly over the entire voltage and temperature ranges expected in operation.

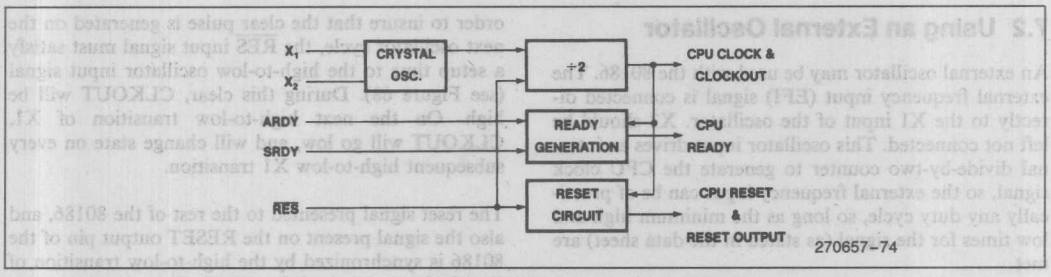
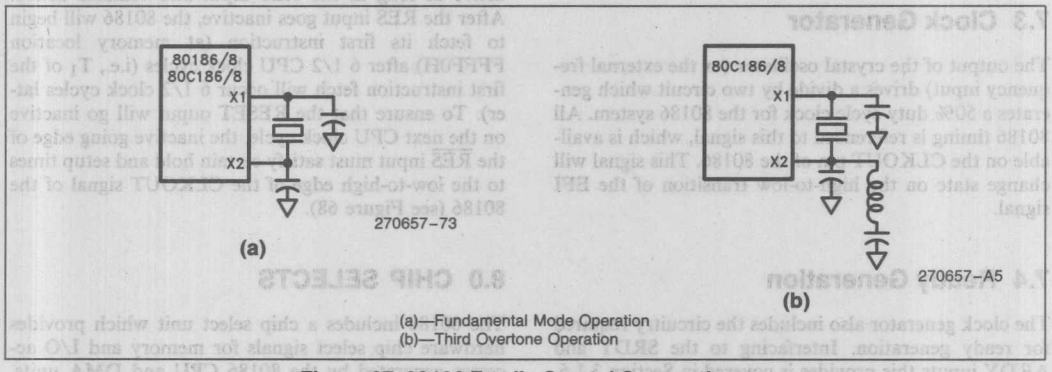
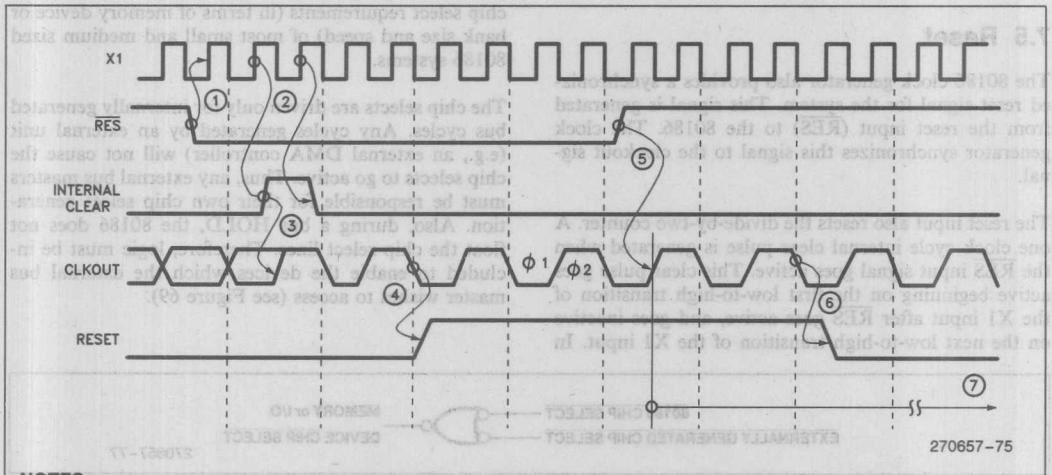


Figure 66. 80186 Clock Generator Block Diagram



(a)—Fundamental Mode Operation
(b)—Third Overtone Operation

Figure 67. 80186 Family Crystal Connections



NOTES:
1. $\overline{RES}$ sampled on rising edge of oscillator input signal X1.
2. Internal clear pulse generated.
3. Internal clear pulse drives CLKOUT high, resynchronizing the clock generator.
4. RESET output goes active (T_{CLRO}).
5. $\overline{RES}$ allowed to go inactive after minimum 4 CLKOUT cycles, recognized at rising CLKOUT.
6. RESET output goes inactive 1½ CLKOUT cycles plus T_{CLRO} after recognition of $\overline{RES}$ inactive.
7. First instruction prefetch occurs 6½ CLKOUT cycles after coming out of reset.

Figure 68. 80186 Clock Generator Reset

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left not connected. This oscillator input drives an internal divide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in Section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input ($\overline{RES}$) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the $\overline{RES}$ input signal goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after $\overline{RES}$ goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next oscillator cycle, the $\overline{RES}$ input signal must satisfy a setup time to the high-to-low oscillator input signal (see Figure 68). During this clear, CLKOUT will be high. On the next high-to-low transition of X1, CLKOUT will go low, and will change state on every subsequent high-to-low X1 transition.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the $\overline{RES}$ input also remains active. After the $\overline{RES}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To ensure that the RESET ouput will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{RES}$ input must satisfy certain hold and setup times to the low-to-high edge of the CLKOUT signal of the 80186 (see Figure 68).

8.0 CHIP SELECTS

The 80186 includes a chip select unit which provides hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 69).

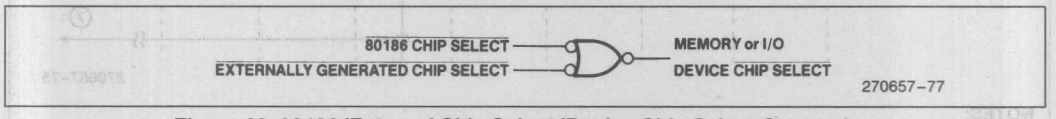


Figure 69. 80186/External Chip Select/Device Chip Select Generation

8.1 Memory Chip Selects

The 80186 provides six discrete memory chip select lines. These signals are named $\overline{UCS}$, $\overline{LCS}$, and $\overline{MCS0\text{-}3}$ for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Select 0–3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 70).

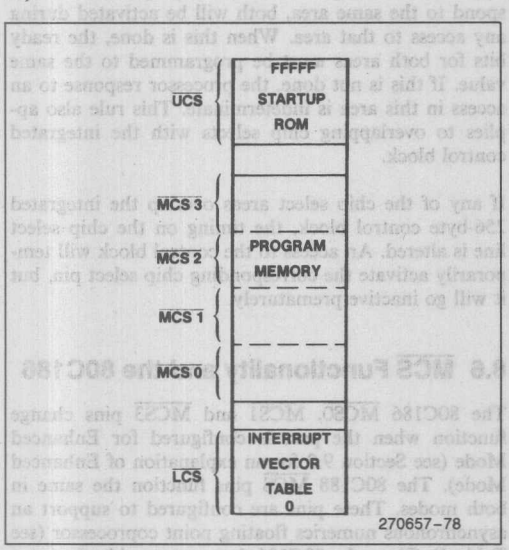


Figure 70. 80186 Memory Areas & Chip Selects

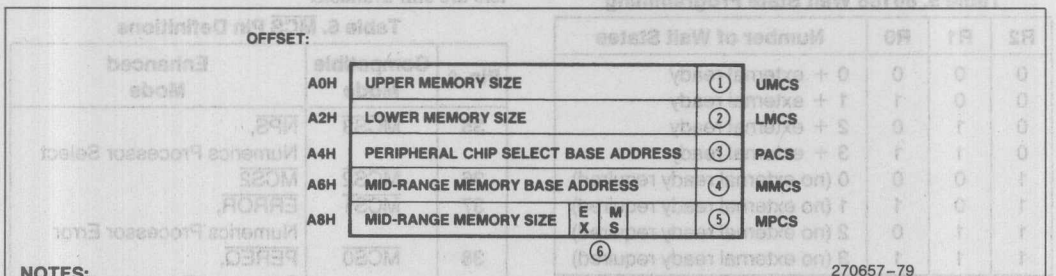
The upper limit of $\overline{UCS}$ and the lower limit of $\overline{LCS}$ are fixed at FFFFFH and 00000H in memory space, respectively. The other limit is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 71). These include 1 each for $\overline{UCS}$ and $\overline{LCS}$, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

On reset, only $\overline{UCS}$ is active. It is programmed to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see Section 8.3 for more information on internal ready generation). None of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. Each of these lines is active for one of seven continuous 128 byte areas in memory or I/O space above a programmed base address.



NOTES:
1. Upper memory ready bits
2. Lower memory ready bits
3. PCS0–PCS3 ready bits
4. Mid-range memory ready bits
5. PCS4–PCS6 ready bits
6. MS: 1 = Peripherals active in memory space
   0 = Peripherals active in I/O space
   EX:1 = 7 PCS lines
   0 = PCS5 = A1, PCS6 = A2

Not all bits of every field are used

Figure 71. 80186 Chip Select Control Registers

The peripheral chip selects are controlled bt two registers in the internal peripheral control block (see Figure 71). These registers set the base address of the peripherals and map the peripherals into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows $\overline{PCS5}$ and $\overline{PCS6}$ to become latched A1 and A2 outputs. When this option is selected, $\overline{PCS5}$ and $\overline{PCS6}$ reflect the state of A1 and A2 throughout a bus cycle. These allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high.

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

For each ready generation area, 0–3 wait states may be inserted by the internal unit. Table 5 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore or include the state of the external ready pins. When using both internal and external ready generation, both elements must be fulfilled before a busy cycle will end. The external ready condition is always required upon RESET for accesses involving the top 1K of memory. Therefore, at least one of the ready pins must be connected to functional ready circuitry or be tied HIGH until $\overline{UCS}$ is reprogrammed early in the initialization sequence.

Table 5. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples using the chip select lines are given in the bus interface section of this note (Section 3.2). The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186. When another master has the bus, it must generate its own chip selects. In addition, whenever the bus is given by the 80186 to an external master (through HOLD/HLDA) the 80186 does **not** float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas **must** be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate. This rule also applies to overlapping chip selects with the integrated control block.

If any of the chip select areas overlap the integrated 256-byte control block, the timing on the chip select line is altered. An access to the control block will temporarily activate the corresponding chip select pin, but it will go inactive prematurely.

8.6 $\overline{MCS}$ Functionality and the 80C186

The 80C186 $\overline{MCS0}$, $\overline{MCS1}$ and $\overline{MCS3}$ pins change function when the part is configured for Enhanced Mode (see Section 9.0 for an explanation of Enhanced Mode). The 80C188 $\overline{MCS}$ pins function the same in both modes. These pins are configured to support an asynchronous numerics floating point coprocessor (see Table 6). Thus, the 80C186 does not provide the complete range of middle chip selects normally available. However, the functionality of the $\overline{MCS2}$ pin and the programming features of the MPCS and MMCS registers are still available.

Table 6. $\overline{MCS}$ Pin Definitions

Pin #	Compatible Mode	Enhanced Mode
35	$\overline{MCS3}$	$\overline{NPS}$, Numerics Processor Select
36	$\overline{MCS2}$	$\overline{MCS2}$
37	$\overline{MCS1}$	$\overline{ERROR}$, Numerics Processor Error
38	$\overline{MCS0}$	$\overline{PEREQ}$, Processor Extension Request

In Enhanced Mode, it is still possible to program the starting address, block size and ready requirements of the middle chip selects. This allows the user to take advantage of the wait-state generation logic on the 80C186 even though the majority of external chip selects are not active. It is also possible to use $\overline{MCS2}$ which is active for one fourth the block size (see Figure 72).

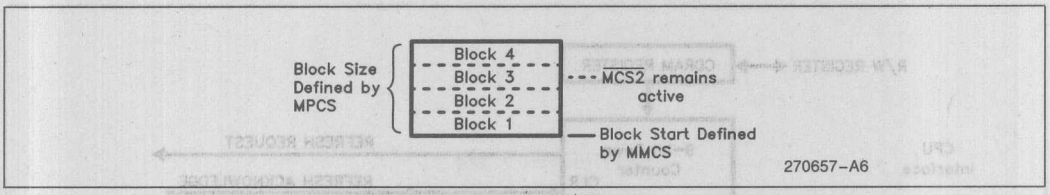


Figure 72. MCS2 Functionality During Enhanced Mode

9.0 80C186 PRODUCT ENHANCEMENTS

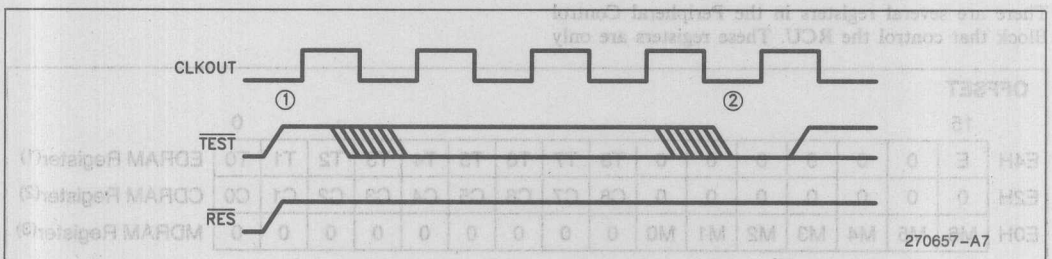
The 80C186 and 80C188 are for the most part identical to their NMOS counterparts, and may be used interchangeably. However, aside from the fact that the 80C186 and 80C188 are designed with Intel's CHMOS III technology and provide greater operating frequencies and less power consumption, they also provide two new operating units not found on the 80186 or 80188: the Refresh Control Unit and the Power-Save Unit. To ensure that the new features of the 80C186 are not accidentally programmed in older designs, the 80C186 has two operating modes: Compatible Mode and Enhanced Mode. Compatible Mode implies that the register, programming and pin definition of the 80C186 is identical to that of the 80186. Enhanced Mode implies that the 80C186 provides a super-set of functionality to that of the 80186.

The different modes are selected during RESET. The timing diagram in Figure 73 shows how the 80C186 samples the TEST input pin just before and just after RES is removed to determine if the device will enter Enhanced Mode. Tying the RESET output pin back to the TEST input pin ensures that the 80C186 or 80C188 enters enhanced mode. If the TEST input is used for external synchronization of code, then RESET can be OR'ed with the other input provided it is always active (low) just after RESET.

When the 80C186 (not the 80C188) is in Enhanced Mode, some of the MCS chip select lines change functionality to support an asynchronous numerics floating-point coprocessor. Refer to Section 8.6 for more detail.

9.1 Refresh Control Unit

To simplify the design of a dynamic memory controller, the 80C186 incorporates integrated address and clock counters which, along with the BIU, facilitate dynamic memory refreshing. A block diagram of the Refresh Control Unit (RCU) and its relationship to the BIU is shown in Figure 74. To the memory interface, a refresh request looks exactly like a memory read bus cycle. This is because a refresh bus cycle is a memory read operation. Because the RCU is integrated into the 80C186, functions such as chip selects and wait-state control can be used effectively.



NOTES:
1. TEST must be high TINVCH when RES transitions high.
2. TEST must be low TINVCH four clocks later.

Figure 73. Enhanced Mode Enable Pin Timing

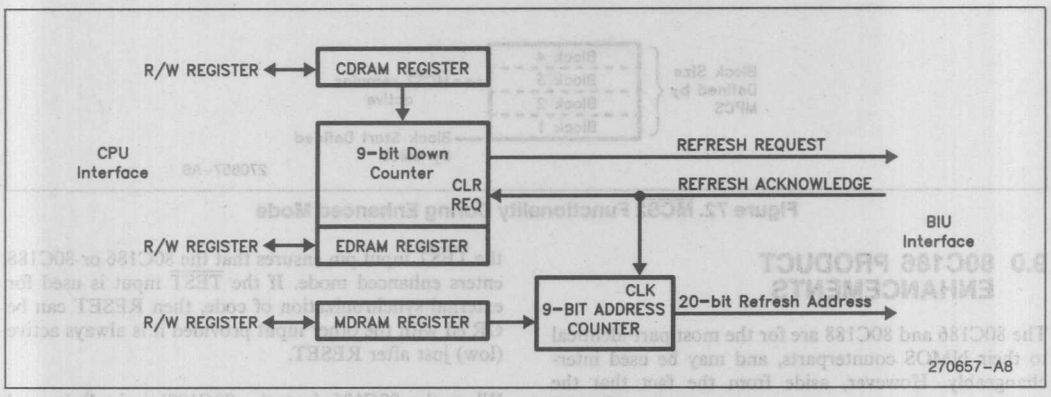


Figure 74. Refresh Control Unit Block Diagram

The 9-bit counter is controlled by the BIU and is used whenever a refresh bus cycle is executed. Thus, any dynamic memory whose refresh address requirement does not exceed nine bits can be directly supported by the 80C186. The 9-bit address counter along with a 6-bit base register define a full 20-bit refresh address. The 9-bit counter generates a signal to initiate a refresh bus cycle. When the counter decrements to 1 (it is decremented every clock cycle), a refresh request is presented to the BIU. When the bus is free, the BIU will run the refresh (memory read) bus cycle. Refresh requests have a higher priority than any other bus request (i.e., CPU, DMA, HOLD).

9.1.1 REFRESH CONTROL UNIT PROGRAMMING

There are several registers in the Peripheral Control Block that control the RCU. These registers are only accessible when the 80C186 or 80C188 are operating in Enhanced mode. Otherwise, a read or write to these registers is ignored.

The three control registers are MDRAM, CDRAM, and EDRAM (see Figure 75). These registers define the operating characteristics of the RCU. The MDRAM register programs the base address (upper 7 bits) of the refresh address (see Figure 76). This allows the refresh address to be mapped into any 4 kilobyte boundary within the 1 megabyte 80C186 address space. The MDRAM register is not altered whenever the refresh address bits (A1 through A9 in Figure 76) roll over. In other words, the refresh address does not act like a linear counter found in a typical DMA controller.

OFFSET

	15															0	
E4H	E	0	0	0	0	0	0	T8	T7	T6	T5	T4	T3	T2	T1	T0	EDRAM Register(1)
E2H	0	0	0	0	0	0	0	C8	C7	C6	C5	C4	C3	C2	C1	C0	CDRAM Register(2)
E0H	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0	MDRAM Register(3)

NOTES:
1. Bits 0–8: T0–T8, Refresh request down counter clock count. These bits are read only and represent the current value of the counter. Any write operation to these bits are ignored.
Bit 15: E, enables the operation of the refresh control unit.
2. Bits 0–8: C0–C8, define the number of CLKOUT cycles between each refresh request.
3. Bits 9–15: M0–M6, are used to define address bits A13–A19 (respectively) of the 20-bit memory address. These bits are set to zero on RESET.

Figure 75. Refresh Control Unit Registers

Address Bit	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Refresh Address	M6	M5	M4	M3	M2	M1	M0	0	0	0	C_{A8}	C_{A7}	C_{A6}	C_{A5}	C_{A4}	C_{A3}	C_{A2}	C_{A1}	C_{A0}	1

Bit 0: Always driven to 1 (Logic High). This is true for both the 80C186 and the 80C188.
Bits 1–9: C_{A0}–C_{A8}, are generated by the 9-bit Linear-Feedback shift counter.
Bits 10–12: Always driven to 0 (Logic Low).
Bits 13–19: M0–M6, are defined by the MDRAM Register.

Figure 76. Physical Address Generation

The CDRAM register defines the time interval between refresh requests by initializing the value loaded into the 9-bit down counter. Thus, the higher the value, the longer the amount of time between requests. The down-counter is decremented every falling edge of CLKOUT, regardless of the activity of the CPU or BIU. When the counter decrements to 1, a request is generated and the counter is again loaded with the value in the CDRAM register. The amount of time between refresh requests can be calculated using the equation shown in Figure 77. The minimum value that can be programmed into the CDRAM register is 18 (12H) regardless of the operating frequency. This is due to the minimum number of clocks required between each successive request to ensure the BIU has enough time to execute the refresh bus cycle. The BIU is not capable of queueing requests; if another request is generated before the current request is executed, the current request is lost. This applies only to the request itself, not the address associated to the request. The refresh address is only changed after the BIU has run the bus cycle. Thus it is possible to miss refresh requests, but not refresh addresses.

The EDRAM register has two functions, depending on whether it is being written or read. During writes to the EDRAM register, only the Enable bit is active. Setting the Enable bit enables the RCU while clearing the Enable bit disables the RCU. Whenever the RCU is enabled, the contents of the CDRAM register are loaded into the 9-bit down counter and refresh requests will be generated when the counter reaches 1. Disabling the RCU stops and clears the counter. A read of the EDRAM register will return the current value of the Enable bit as well as the current value of the 9-bit down counter (zero if the RCU is not enabled). Writing to EDRAM register when the RCU is running does not modify the count value in the 9-bit counter.

9.1.2 REFRESH CONTROL UNIT OPERATION

Figure 78 illustrates the two major functions of the refresh control unit that are responsible for initiating and controlling the refresh bus cycles.

The down counter is loaded on the falling edge of CLKOUT, when either the Enable bit is set or the counter decrements to 1. Once loaded, the down counter will decrement every falling edge of CLKOUT (as long as the Enable bit remains set).

When the counter decrements to 1, two things happen. First, a request is generated to the BIU to run a refresh bus cycle. The request remains active until the bus cycle is run. Second, the down counter is reloaded with the value contained in the CDRAM register. At this time, the down counter will again begin counting down every clock cycle. It does not wait until the request has been serviced. This is done to ensure that each refresh request occurs at the correct interval. Otherwise, the time between refresh requests would also be a function of bus activity, which is unpredictable. When the BIU services the refresh request, it will clear the request and increment the refresh address.

$$\frac{R_{PERIOD}\ (\mu s) * FREQ\ (MHz)}{\#\text{Refresh Rows} + (\#\ \text{Refresh Rows} * \%\ \text{Overhead})} = \text{CDRAM Register Value}$$

R_{Period} = Maximum Refresh period specified by the DRAM manufacturer (time in microseconds).
FREQ = Operating Frequency at 80C186 in MHz.
# Refresh Rows = Total number of rows to be refreshed.
% Overhead = Derating factor to compensate for missed refresh requests (typically 1–5%).

Figure 77. Equation to Calculate Refresh Interval

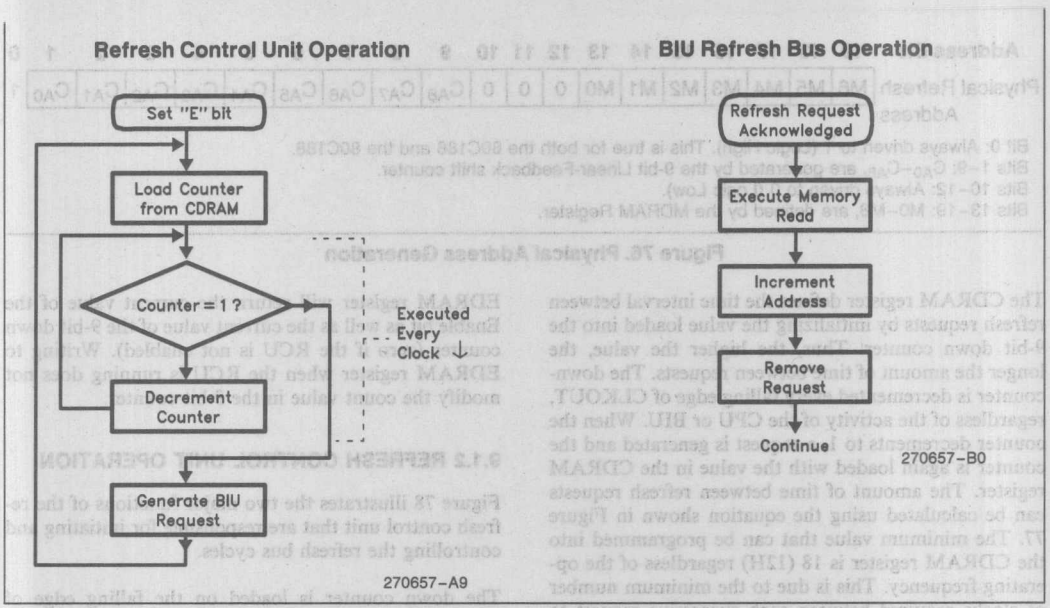


Figure 78. Flowchart of RCU Operation

9.1.3 REFRESH ADDRESS CONSIDERATIONS

The physical address that is generated during a refresh bus cycle is shown in Figure 76, and applies to both the 80C186 and 80C188. The refresh address bits C_{A0} through C_{A8} are generated using a linear-feedback shift counter which does not increment the addresses linearly from 0 through 1FFH (although they do follow a predictable algorithm). Further, note that for the 80C188, address bit A0 does not toggle during refresh operation, which means that it cannot be used as part of the refresh address applied to the dynamic memory device. Typically, A0 is used as part of memory decoding in 80C188 applications, unlike the 80C186 which uses A0 along with $\overline{BHE}$ to select an upper or lower bank. Therefore, when designing with the 80C188, it is important **not** to include A0 as part of the row address that is used for refreshing. Appendix K illustrates memory address multiplexing techniques that can be applied to the 80C186 and 80C188.

9.1.4 REFRESH OPERATION AND BUS HOLD

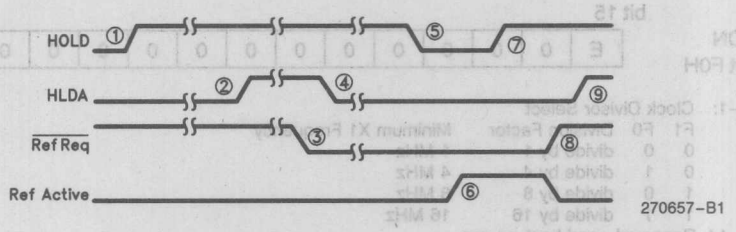
When another bus master has control of the bus, the HLDA signal is kept active as long as the HOLD input remains active. If a refresh request is generated while HOLD is active, the 80C186 will remove (drive inactive) the HLDA signal to indicate to the current bus master that the 80C186 wishes to regain control of the bus (see Figure 79). Only when the HOLD input is removed will the BIU begin the refresh bus cycle.

Therefore, it is the responsibility of the system designer to ensure that the 80C186 can regain the bus if a refresh request is signalled. The sequence of HLDA going inactive while HOLD is active can be used to signal a pending refresh request. HOLD need only go inactive for one clock period to allow the refresh bus cycle to be run. If HOLD is again asserted, the 80C186 will give up the bus after the refresh bus cycle has been run (provided there is not another refresh request generated during that time).

9.2 Power-Save Unit

The Power-Save Unit is intended to benefit applications by lower power consumption while maintaining regular operation of the CPU. The 80C186 Power-Save mechanism lowers current needs by reducing the operating frequency.

The Power-Save Unit is an internal clock divider as shown in Figure 80. Because the Power-Save Unit will change the internal operating frequency, all other units within the 80C186 will be affected by the clock change. This includes the CPU, Timers, Refresh, DMA, and BIU. Thus, by using the Power-Save feature, the net effect is similar to changing the input clock frequency.



NOTES:
1. System generates HOLD request.
2. HLDA is returned and 80C186 floats the bus/control lines.
3. Refresh request is generated by the 80C186.
4. 80C186 lowers (removes) HLDA to signal that it wants the bus back.
5. 80C186 waits until HOLD is lowered (removed) for at least 1 clock cycle (minimum HOLD setup and hold time) to execute the refresh bus cycle. If HOLD is never lowered, the 80C186 will not take over the bus.
6. 80C186 runs the refresh bus cycle.
7. HOLD can be again asserted after the 1 clock duration.
8. The refresh request is cleared after the bus cycle has been executed.
9. If HOLD was again asserted, the 80C186 will immediately relinquish the bus back. If no HOLD occurred, normal CPU operation will resume.

Figure 79. HOLD/HLDA Timing and Refresh Request

9.2.1 POWER-SAVE UNIT PROGRAMMING

The PDCON register (see Figure 81) controls the operation of the Power-Save Unit. This register is available for programming when the 80C186 or 80C188 is in Enhanced Mode. Reads or write to the PDCON register in Compatible Mode result in no operation, and the value returned will be all ones.

When the Enable bit in the PDCON register is set, the Power-Save Unit is active and, depending on the condition of the F0 and F1 bits, the operating clock of the 80C186 is changed from normal operation. When the Enable bit is cleared, the 80C186 will operate at the standard divide by 2 clock rate. The Enable bit is automatically cleared whenever a non-masked interrupt occurs. Thus, if the Power-Save feature is enabled and an unmasked interrupt of sufficient priority is received, the Enable bit clears and the processor executes at full speed. This allows interrupts to be processed at full speed. A return from the interrupt does not automatically set the Enable bit. This must be done as part of the interrupt routine. Software interrupts do not clear the Enable bit.

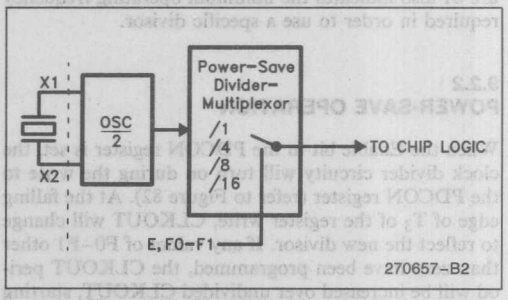
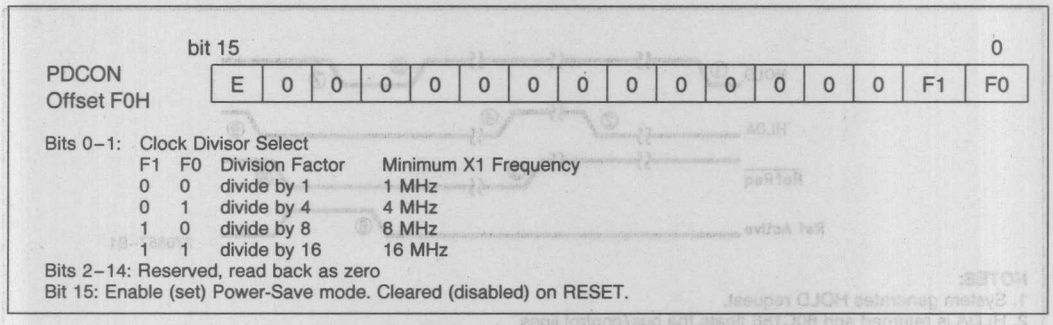


Figure 80. Simplified Power-Save Internal Operation



Bits 0–1: Clock Divisor Select

F1	F0	Division Factor	Minimum X1 Frequency
0	0	divide by 1	1 MHz
0	1	divide by 4	4 MHz
1	0	divide by 8	8 MHz
1	1	divide by 16	16 MHz

Bits 2–14: Reserved, read back as zero
Bit 15: Enable (set) Power-Save mode. Cleared (disabled) on RESET.

Figure 81. Power-Save Register Definition

The F0 and F1 bits determine the divisor of the Power-Save unit. Figure 81 provides a list of the various combinations of the bits and their division factor. Note that the divisor is related to the output clock, not the input clock at pin X1. Selecting a divisor of 1 does not reduce the power consumption. The operating clock of the 80C186 must not be divided below the minimum operating frequency specified in data sheet (500 kHz). Figure 81 also indicates the minimum operating frequency required in order to use a specific divisor.

9.2.2 POWER-SAVE OPERATION

When the Enable bit in the PDCON register is set, the clock divider circuity will turn on during the write to the PDCON register (refer to Figure 82). At the falling edge of T_3 of the register write, CLKOUT will change to reflect the new divisor. If any values of F0–F1 other than zero have been programmed, the CLKOUT period will be increased over undivided CLKOUT, starting with the low phase. CLKOUT will not glitch.

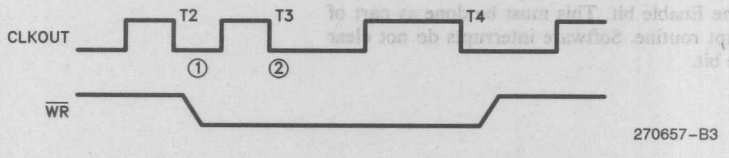
The Power-Save Unit remains active until one of three events happens: either the Enable bit in the PDCON register is cleared, new values for F0 and F1 are programmed, or an unmasked interrupt is received. In the first two cases, the changes directly follow Figure 82. When an unmasked interrupt is received, the operating frequency is changed as shown in Figure 82, but may occur at **any** T_3 bus state in progress at the time of the interrupt. Thus, it is not possible to determine exactly when, in the event of an interrupt, the Power-Save unit will be disabled.

10.0 SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

10.1 System Initialization in an 80186 System

The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals can be used.



NOTES:
1. Write to PDCON register (as viewed on the bus)
2. Low going edge of T3 starts new clock divider

Figure 82. Power-Save Clock Transition

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs the processor will fetch garbage. Appendix F shows a typical initialization sequence for the 80186 chip select unit.

10.2 Instruction Execution Differences between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

UNDEFINED OPCODES:

When the opcodes 63H, 64H, 65H, 66H, 67H, F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH OPCODE:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will excecute an illegal instruction exception, interrupt type 6.

WORD WRITE AT OFFSET FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

SHIFT/ROTATE BY VALUE GREATER THAN 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK PREFIX:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

NOTE:
When executing more than one LOCKed instruction, always make sure there are 6 bytes of code between the end of the first LOCKed instruction and the start of the second LOCKed instruction.

INTERRUPTED STRING MOVE INSTRUCTIONS:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be reexecuted upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

CONDITIONS CAUSING DIVIDE ERROR WITH AN INTEGER DIVIDE:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater than 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling −32768 and −128 in decimal, respectively).

ESC OPCODE:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for coprocessors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in Appendix H of this note.

APPENDIX A
PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers encompasses 256 contiguous bytes and can be located on any 256 byte boundary of the 80186 memory or I/O space. Maps of these registers are shown in Figure A-1 for the 80186/80188 and in Figure A-2 for the 80C186/80C188. Any unused bytes are reserved.

A.1 SETTING THE BASE LOCATION OF THE PERIPHERAL CONTROL BLOCK

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation registers will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into slave mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

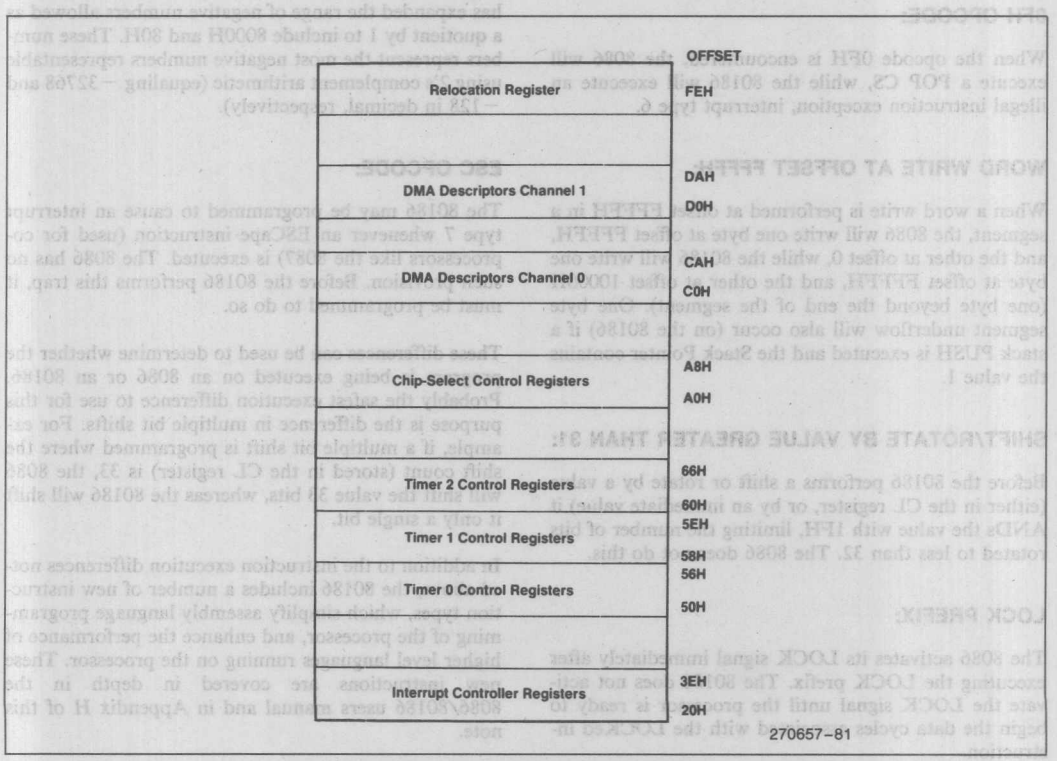


Figure A-1. 80186/80188 Integrated Peripheral Control Block

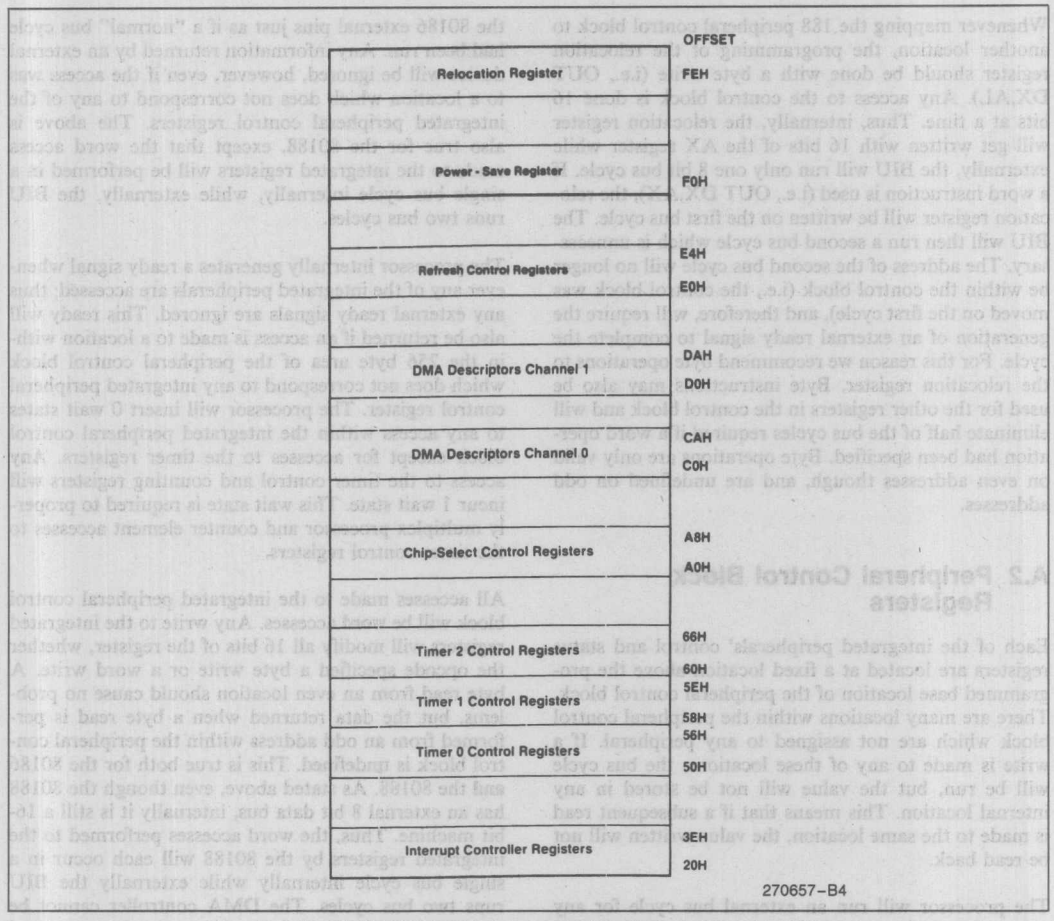
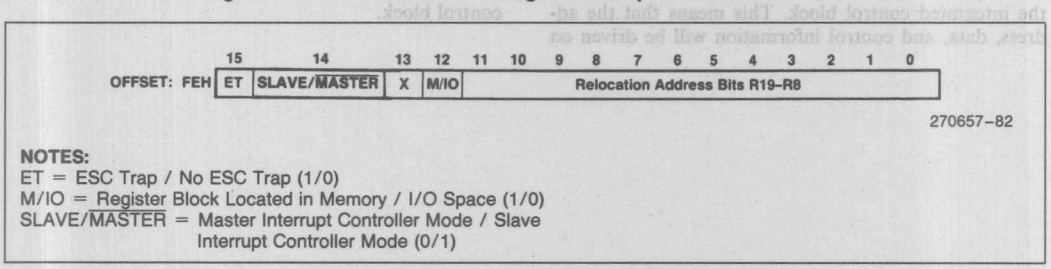


Figure A-2. 80C186/80C188 Integrated Peripheral Control Block



NOTES:
ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register Block Located in Memory / I/O Space (1/0)
SLAVE/MASTER = Master Interrupt Controller Mode / Slave Interrupt Controller Mode (0/1)

Figure A-3. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEH in I/O space.

To relocate the peripheral control block to the memory range 10000H-100FFH, for example, the user programs the relocation register with the value 1100H. Since the relocation register is contained within the peripheral control block, it moves to word location 100FEH in memory space.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e., OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e., OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e., the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle internally, while externally, the BIU runs two bus cycles.

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored. This ready will also be returned if an access is made to a location within the 256 byte area of the peripheral control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. **Any** access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block will be **word** accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 **and** the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16-bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle internally while externally the BIU runs two bus cycles. The DMA controller cannot be used for either read or write accesses to the peripheral control block.

APPENDIX B
80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 WHY SYNCHRONIZERS ARE REQUIRED

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch long enough for the output to stabilize, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have.

Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

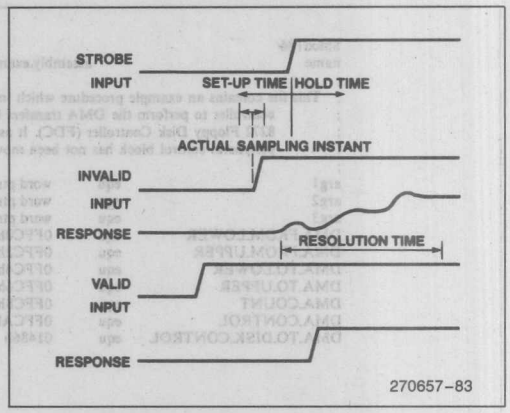


Figure B-1. Valid and Invalid Latch Input Transitions and Responses

B.2 80186 SYNCHRONIZERS

The 80186 contains synchronizers on the $\overline{RES}$, $\overline{TEST}$, TmrIn0–1, DRQ0–1, NMI, INT0–3, ARDY, and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C
80186 EXAMPLE DMA INTERFACE CODE

```
$mod186
name                    assembly.example.80186.DMA_support
;
;  This file contains an example procedure which initializes the 80186 DMA
;       controller to perform the DMA transfers between the 80186 system and the
;       8272 Floppy Disk Controller (FDC). It assumes that the 80186
;       peripheral control block has not been moved from its reset location.
;
arg1                    equ     word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
DMA_FROM_LOWER          equ     0FFC0h          ;  DMA register locations
DMA_FROM_UPPER          equ     0FFC2h
DMA_TO_LOWER            equ     0FFC4h
DMA_TO_UPPER            equ     0FFC6h
DMA_COUNT               equ     0FFC8h
DMA_CONTROL             equ     0FFCAh
DMA_TO_DISK_CONTROL     equ     01486h          ;  destination synchronization
                                                ;  source to memory, incremented
                                                ;  destination to I/O
                                                ;  no terminal count
                                                ;  byte transfers

DMA_FROM_DISK_CONTROL equ       0A046h          ;  source synchronization
                                                ;  source to I/O
                                                ;  destination to memory, incr
                                                ;  no terminal count
                                                ;  byte transfers
FDC_DMA                 equ     6B8h            ;  FDC DMA address
FDC_DATA                equ     688h            ;  FDC data register
FDC_STATUS              equ     680h            ;  FDC status register

cgroup                  group   code
code                    segment                 public 'code'
                        public  set_dma_
                        assume  cs:cgroup
;
;  set_dma (offset,to) programs the DMA channel to point one side to the
;       disk DMA address, and the other to memory pointed to by ds:offset. If
;       'to' = 0 then will be a transfer from disk to memory; if
;       'to' = 1 then will be a transfer from memory to disk. The parameters to
;       the routine are passed on the stack.
;
set_dma_                proc    near
                        enter   0,0             ;  set stack addressability
                        push    AX              ;  save registers used
                        push    BX
                        push    DX
                        test    arg2,1          ;  check to see direction of
                                                ;  transfer
                        jz      from_disk
;  performing a transfer from memory to the disk controller
;
                        mov     AX,DS           ;  get the segment value
                        rol     AX,4            ;  gen the upper 4 bits of the
                                                ;  physical address in the lower 4
                                                ;  bits of the register
```

270657-84

```
                        mov     BX,AX                   ; save the result...
                        mov     DX,DMA_FROM_UPPER       ; prgm the upper 4 bits of the
                        out     DX,AX                   ; DMA source register
                        and     AX,0FFF0h               ; form the lower 16 bits of the
                                                        ; physical address
                        add     AX,arg1                 ; add the offset
                        mov     DX,DMA_FROM_LOWER       ; prgm the lower 16 bits of the
                        out     DX,AX                   ; DMA source register
                        jnc     no_carry_from           ; check for carry out of addition
                        inc     BX                      ; if carry out, then need to adj
                        mov     AX,BX                   ; the upper 4 bits of the pointer
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
no_carry_from:
                        mov     AX,FDC_DMA              ; prgm the low 16 bits of the DMA
                        mov     DX,DMA_TO_LOWER         ; destination register
                        out     DX,AX
                        xor     AX,AX                   ; zero the up 4 bits of the DMA
                        mov     DX,DMA_TO_UPPER         ; destination register
                        out     DX,AX
                        mov     AX,DMA_TO_DISK_CONTROL; prgm the DMA ctl reg
                        mov     DX,DMA_CONTROL          ; note: DMA may begin immediatly
                        out     DX,AX                   ; after this word is output
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
from_disk:
;
;   performing a transfer from the disk to memory
;
                        mov     AX,DS
                        rol     AX,4
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
                        mov     BX,AX
                        and     AX,0FFF0h
                        add     AX,arg1
                        mov     DX,DMA_TO_LOWER
                        out     DX,AX
                        jnc     no_carry_to
                        inc     BX
                        mov     AX,BX
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
no_carry_to:
                        mov     AX,FDC_DMA

                        mov     DX,DMA_FROM_LOWER
                        out     DX,AX
                        xor     AX,AX
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
                        mov     AX,DMA_FROM_DISK_CONTROL
                        mov     DX,DMA_CONTROL                                  270657-85
                        out     DX,AX
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
set_dma_                endp

code                    ends
                        end
```

270657-86

APPENDIX D
80186 EXAMPLE TIMER INTERFACE CODE

```
$mod186
name                                    example_80186_timer_code
;
;  this file contains example 80186 timer routines. The first routine
;       sets up the timer and interrupt controller to cause the timer
;       to generate an interrupt every 10 milliseconds, and to service
;       interrupt to implement a real time clock. Timer 2 is used in
;       this example because no input or output signals are required.
;       The code example assumes that the peripheral control block has
;       not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1                    equ     word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
timer_2int              equ     19                      ;  timer 2 has vector type 19
timer_2control          equ     0FF66h
timer_2max_ctl          equ     0FF62h
timer_int_ctl           equ     0FF32h                  ;  interrupt controller regs
eoi_register            equ     0FF22h
interrupt_stat          equ     0FF30h

data                    segment                         public 'data'
                        public  hour_,minute_,second_,msec_
msec_                   db      ?
hour_                   db      ?
minute_                 db      ?
second_                 db      ?
data                    ends

cgroup                  group   code
dgroup                  group   data

code                    segment                         public 'code'
                        public  set_time_
                        assume  cs:code,ds:dgroup
;
;  set_time(hour,minute,second) sets the time variables, initializes the
;       80186 timer2 to provide interrupts every 10 milliseconds, and
;       programs the interrupt vector for timer 2
;
set_time_               proc    near
                        enter   0,0                     ;  set stack addressability
                        push    AX                      ;  save registers used
                        push    DX
                        push    SI
                        push    DS

                        xor     AX,AX                   ;  set the interrupt vector
                                                        ;  the timers have unique
                                                        ;  interrupt
                                                        ;  vectors even though they share
                                                        ;  the same control register
                        mov     DS,AX

                        mov     SI,4 * timer2_int
```

270657-87

```
                        mov     word ptr DS:[SI],offset timer_2_interrupt_routine
                        inc     SI
                        inc     SI
                        mov     DS:[SI],CS
                        pop     DS

                        mov     AX,arg1                 ; set the time values
                        mov     hour_,AL
                        mov     AX,arg2
                        mov     minute_,AL
                        mov     AX,arg3
                        mov     second_,AL
                        mov     msec_,0

                        mov     DX,timer2_max_ctl       ; set the max count value
                        mov     AX,20000                ; 10 ms / 500 ns (timer 2 counts
                                                        ; at 1/4 the CPU clock rate)
                        out     DX,AX
                        mov     DX,timer2_control       ; set the control word
                        mov     AX,1111000000000001b    ; enable counting
                                                        ; generate interrupts on TC
                                                        ; continuous counting
                        out     DX,AX

                        mov     DX,timer_int_ctl        ; set up the interrupt controller
                        mov     AX,0000b                ; unmask interrupts
                                                        ; highest priority interrupt
                        out     DX,AX
                        sti                             ; enable processor interrupts

                        pop     SI
                        pop     DX
                        pop     AX
                        leave
                        ret
set_time_               endp

timer2_interrupt_routine proc   far
                        push    AX
                        push    DX

                        cmp     msec_,99                ; see if one second has passed
                        jae     bump_second             ; if above or equal...
                        inc     msec_
                        jmp     reset_int_ctl
bump_second:
                        mov     msec_,0                 ; reset millisecond
                        cmp     second_,59              ; see if one minute has passed
                        jae     bump_minute
                        inc     second_
                        jmp     reset_int_ctl
bump_minute:
                        mov     second_,0
                        cmp     minute_,59              ; see if one hour has passed
                        jae     bump_hour
                        inc     minute_
                        jmp     reset_int_ctl
                        pop     DX
                        pop     AX
                        ret
timer2_interrupt_routine endp
code                    ends
                        end
```

270657-88

270657-89

```
bump_hour:
                        mov     minute_,0
                        cmp     hour_,12        ; see if 12 hours have passed
                        jae     reset_hour
                        inc     hour_
                        jmp     reset_int_ctl
reset_hour:
                        mov     hour_,1
reset_int_ctl:

                        mov     DX,eoi_register
                        mov     AX,8000h        ; non-specific end of interrupt
                        out     DX,AX

                        pop     DX
                        pop     AX
                        iret
timer2_interrupt_routine endp
code                    ends
                        end
$mod186
name                    example_80186_baud_code
;
; this file contains example 80186 timer routines. The second routine
;       sets up the timer as a baud rate generator. In this mode,
;       Timer 1 is used to continually output pulses with a period of
;       6.5 usec for use with a serial controller at 9600 baud
;       programmed in divide by 16 mode (the actual period required
;       for 9600 baud is 6.51 usec). This assumes that the 80186 is
;       running at 8 MHz. The code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah

code                    segment                         public 'code'
                        assume  cs:code
;
; set_baud() initializes the 80186 timer1 as a baud rate generator for
;       a serial port running at 9600 baud
;
set_baud_               proc    near
                        push    AX              ; save registers used
                        push    DX

                        mov     DX,timer1_max_cnt ; set the max count value
                        mov     AX,13           ; 500ns * 13 = 6.5 usec
                        out     DX,AX
                        mov     DX,timer1_control ; set the control word
                        mov     AX,1100000000000001b ; enable counting
                                                ; no interrupt on TC
                                                ; continuous counting
                                                ; single max count register
                        out     DX,AX

                        pop     DX
                        pop     AX
```

270657-90

```
                        ret
set_baud_               endp
code                    ends
                        end

$mod186
name                    example.80186.count_code
;
;  this file contains example 80186 timer routines. The third routine
;       sets up the timer as an external event counter. In this mode,
;       Timer 1 is used to count transitions on its input pin. After
;       the timer has been set up by the routine, the number of
;       events counted can be directly read from the timer count
;       register at location FF58H in I/O space. The timer will
;       count a maximum of 65535 timer events before wrapping
;       around to zero. This code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah
timer1_cnt_reg          equ     0FF58H

code                    segment                 public 'code'
                        assume  cs:code
;
;  set_count() initializes the 80186 timer1 as an event counter
;
set_count_              proc    near
                        push    AX                      ; save registers used
                        push    DX

                        mov     DX,timer1_max_cnt       ; set the max count value
                        mov     AX,0                    ; allows the timer to count
                                                        ; all the way to FFFFH
                        out     DX,AX
                        mov     DX,timer1_control       ; set the control word
                        mov     AX,1100000000000101b    ; enable counting
                                                        ; no interrupt on TC
                                                        ; continuous counting
                                                        ; single max count register
                                                        ; external clocking
                        out     DX,AX

                        xor     AX,AX                   ; zero AX
                        mov     DX,timer1_cnt_reg       ; and zero the count in the timer
                        out     DX,AX                   ; count register

                        pop     DX
                        pop     AX
                        ret

set_count_              endp
code                    ends
                        end
```

270657-91

APPENDIX E
80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```
$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller to provide
;       two cascaded interrupt inputs (through an external 8259A
;       interrupt controller on pins INT0/INT2) and two direct
;       interrupt inputs (on pins INT1 and INT3). The default priority
;       levels are used. Because of this, the priority level programmed
;       into the control register is set the 111, the level all
;       interrupts are programmed to at reset.
;
int0_control            equ     0FF38H
int_mask                equ     0FF28H
;
code                    segment                 public 'code'
                        assume  CS:code
set_int_                proc    near
                        push    DX
                        push    AX

                        mov     AX,0100111B     ;  Cascade Mode
                        ;  interrupt unmasked
                        mov     DX,int0_control
                        out     DX,AX

                        mov     AX,01001101B    ;  now unmask the other external
                        ;  interrupts
                        mov     DX,int_mask
                        out     DX,AX
                        pop     AX
                        pop     DX
                        ret
set_int_                endp
code                    ends
                        end

$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller into Slave
;       Mode. This code does not initialize any of the 80186
;       integrated peripheral control registers, nor does it initialize
;       the external 8259A interrupt conroller.
;
relocation_reg          equ     0FFFEH
;
code                    segment                 public 'code'
                        assume  CS:code
set_Slave               proc    near
                        push    DX
                        push    AX

                        mov     DX,relocation_reg
                        in      AX,DX           ;  read old contents of register
                        or      AX,0100000000000000B    ;  set the Slave/Master mode bit
                        out     DX,AX
```

270657-92

APPENDIX F
80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```
name            example_80186_system_init
;
; This file contains a system initialization routine for the 80186
;       or the 8086. The code determines whether it is running on
;       an 80186 or an 8086, and if it is running on an 80186, it
;       initializes the integrated chip select registers.
;
restart         segment at              0FFFFh
;
; This is the processor reset address at 0FFFF0H
;
                org     0
                jmp     far ptr initialize
restart         ends
;
                extrn   monitor:far
init_hw         segment at              0FFF0h
                assume  CS:init_hw
;
; This segment initializes the chip selects. It must be located in the
;       top 1K to insure that the ROM remains selected in the 80186
;       system until the proper size of the select area can be programmed.
;
UMCS_reg        equ     0FFA0H          ; chip select register locations
LMCS_reg        equ     0FFA2H
PACS_reg        equ     0FFA4H
MPCS_reg        equ     0FFA8H
UMCS_value      equ     0F038H          ; 64K, no wait states
LMCS_value      equ     07F8H           ; 32K, no wait states
PACS_value      equ     007EH           ; peripheral base at 400H, 2 ws
MPCS_value      equ     81B8H           ; PCS5 and 6 supplies,
                                        ; peripherals in I/O space
initialize      proc    far
                mov     AX,2            ; determine if this is an
                mov     CL,33           ; 8086 or an 80186 (checks
                shr     AX,CL           ; to see if the multiple bit
                test    AX,1            ; shift value was ANDed)
                jz      not_80186

                mov     DX,UMCS_reg     ; program the UMCS register
                mov     AX,UMCS_value
                out     DX,AX

                mov     DX,LMCS_reg     ; program the LMCS register
                mov     AX,LMCS_value
                out     DX,AX

                mov     DX,PACS_reg     ; set up the peripheral chip
                                        ; selects (note the mid-range
                                        ; memory chip selects are not
                                        ; needed in this system, and
                                        ; are thus not initialized
                mov     AX,PACS_value
                out     DX,AX                                   270657-93
                mov     DX,MPCS_reg
                mov     AX,MPCS_value
                out     DX,AX
;
; Now that the chip selects are all set up, the main program of the
;       computer may be executed.
;
not_80186:
                jmp     far ptr monitor
initialize      endp
init_hw         ends
                end
                                        270657-94
```

APPENDIX G
80186 WAIT STATE PERFORMANCE

Because the 80186 contains separate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performance degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not affect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution time calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot access the opcode bytes immediately upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1		Program 2	
	Exec Time (μsec)	Perf Degr	Exec Time (μsec)	Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```
$mod186
name                    example_wait_state_performance
;
; This file contains two programs which demonstrate the 80186 performance
;       degradation as wait states are inserted. Program 1 performs a
;       transformation between two types of characters sets, then copies
;       the transformed characters back to the original buffer (which is 64
;       bytes long. Program 2 performs the same type of transformation, however
;       instead of performing a table lookup, it multiplies each number in the
;       original 32 word buffer by a constant (3, note the use of the integer
;       immediate multiply instruction). Program "nothing" is used to measure
;       the call and return times from the driver program only.
;
cgroup                  group   code
dgroup                  group   data
data                    segment                 public 'data'
```

270657-95

```
t_table         db      256 dup (?)
t_string        db      64 dup (?)
m_array         dw      32 dup (?)
data            ends

code            segment                         public 'code'
                assume  CS:cgroup,DS:dgroup
                public  bench_1,bench_2,nothing_,wait_state_,set_timer_
bench_1         proc    near
                push    SI              ;       save registers used
                push    CX
                push    BX
                push    AX

                mov     CX,64           ;       translate 64 bytes
                mov     SI,0
                mov     BH,0

loop_back:
                mov     BL,t_string[SI] ;       get the byte
                mov     AL,t_table[BX]  ;       translate byte
                mov     t_string[SI],AL ;       and store it
                inc     SI              ;       increment index
                loop    loop_back       ;       do the next byte

                pop     AX
                pop     BX
                pop     CX
                pop     SI
                ret
bench_1         endp

bench_2         proc    near
                push    AX              ;       save registers used
                push    SI
                push    CX

                mov     CX,32           ;       multiply 32 numbers
                mov     SI,offset m_array

loop_back_2:
                imul    AX,word ptr [SI],3      ;       immediate multiply
                mov     word ptr [SI],AX
                inc     SI
                inc     SI
                loop    loop_back_2

                pop     CX
                pop     SI
                pop     AX
                ret
bench_2_        endp
```

270657-96

```
nothing_        proc    near
                ret
nothing_        endp
;
;  wait_state(n) sets the 80186 LMCS register to the number of wait states
;       (0 to 3) indicated by the parameter n (which is passed on the stack).
;       No other bits of the LMCS register are modified.
;
wait_state_     proc    near
                enter   0,0             ;  set up stack frame
                push    AX              ;  save registers used
                push    BX
                push    DX

                mov     BX,word ptr [BP + 4]    ;  get argument
                mov     DX,0FFA2h       ;  get current LMCS register
contents
                in      AX,DX

                and     AX,0FFFCh       ;  and off existing ready bits
                and     BX,3            ;  insure ws count is good
                or      AX,BX           ;  adjust the ready bits
                out     DX,AX           ;  and write to LMCS

                pop     DX
                pop     BX
                pop     AX
                leave                   ;  tear down stack frame
                ret
wait_state_     endp
;
;  set_timer() initializes the 80186 timers to count microseconds. Timer 2
;       is set up as a prescaler to timer 0, the microsecond count can be read
;       directly out of the timer 0 count register at location FF50H in I/O
;       space.
;
set_timer_      proc    near
                push    AX
                push    DX

                mov     DX,0ff66h       ;  stop timer 2
                mov     AX,4000h
                out     DX,AX

                mov     DX,0ff50h       ;  clear timer 0 count
                mov     AX,0
                out     DX,AX

                mov     DX,0ff52h       ;  timer 0 counts up to 65535
                mov     AX,0
                out     DX,AX
```

270657-97

```
                        mov     DX,0ff56h       ; enable timer 0
                        mov     AX,0c009h
                        out     DX,AX

                        mov     DX,0ff60h       ; clear timer 2 count
                        mov     AX,0
                        out     DX,AX

                        mov     DX,0ff62h       ; set maximum count of timer 2
                        mov     AX,2
                        out     DX,AX

                        mov     DX,0ff66h       ; re-enable timer 2
                        mov     AX,0c001h
                        out     DX,AX

                        pop     DX
                        pop     AX
                        ret
set_timer_              endp
code                    ends
                        end
```

270657-98

APPENDIX H
80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "$mod186" switch must be given to the assembler. This can be done by placing the line: "$mod186" at the beginning of the assembly language file.

PUSH IMMEDIATE

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

PUSHA, POPA

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DC, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

IMUL BY AN IMMEDIATE VALUE

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

SHIFTS/ROTATES BY AN IMMEDIATE VALUE

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e., the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

BLOCK INPUT/OUTPUT

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory. Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

```
INSB          ;perform byte input

REP OUTSW     ;perform word block output
```

BOUND

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediately the memory location containing the lower bound of the array.

ENTER AND LEAVE

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP              /*save the previous  frame
                     pointer*/
if level=0 then
   BP:=SP;
else   templ:=SP;/*save current frame  pointer
                     */
  temp2:= level - 1;
  do while temp2>0/*copy down previous  level
                     frame*/
    BP:= BP - 2;     /*pointers*/
    PUSH [BP];
  BP:=templ;
  PUSH BP;           /*put current level  frame
                     pointer*/

/*in the save area*/
SP:=SP - disp;       /*create space on the  stack
                     for*/

/*local variables*/
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction.

As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

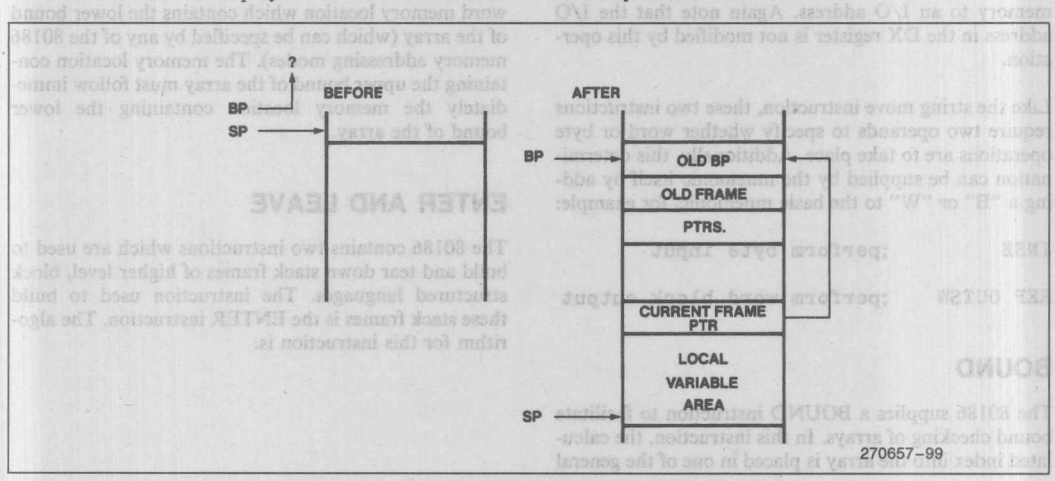


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I
80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8–AD15 on the 80186 are transformed to A8–A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{\text{BHE}}$/S7 is always defined HIGH by the 80188, since the upper half of the data bus is non-existent.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{\text{B}}$/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, not in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When a word access is made to the internal registers, the BIU will run two bus cycles externally.

Access to the control block may also be done with byte operations. Internally the full 16-bits of the AX register will be written, while externally, only one bus cycle will be executed.

APPENDIX J
80186/80C186 DIFFERENCES

There are two operating modes of the 80C186 and 80C188: Compatible Mode and Enhanced Mode. In Compatible Mode, the 80C186 will function identically to the 80186 with the following noted exceptions:

1) All non-initialized registers in the peripheral control block will reset to a random value on power-up on the 80C186. Non-Initialized registers consist of those registers which are not used for control, i.e., address pointers, max count, etc. For compatibility, all registers should be programmed before being used on existing 80186 applications as well as on new 80C186 applications.
2) The ET (Esc/Trap) bit in the relocation register has no effect in Compatible Mode. If an escape opcode is executed, the 80C186 will always trap to an interrupt vector type 7. The 80C186 does not support any numerics operations when in Compatible Mode.

In Enhanced Mode, the 80C186 provides additional features not found on the 80186. There are newly defined registers to support these new features, and three of the output pins of the 80C186 change functionality. The new registers and pin descriptions are covered in Section 9.0.

The 80C188 in Enhanced Mode functions similarly to the 80C186 except for numerics operation. It is not possible to interface a numerics coprocessor with the 80C188. Therefore, none of the $\overline{MCS}$ pins change functionality when invoking Enchanced Mode on the 80C188. Further, any attempted execution of an escape opcode will result in a trap to interrupt vector type 7.

APPENDIX K
DRAM ADDRESSING CONFIGURATIONS FOR THE 80C186/80C188

80C186 DESIGNS

		Row Address (A0–AX)	Column Address (A0–AX)
64K x 1	(128K Bytes)	A1–A8	A9–A16
16K x 4	(32K Bytes)	A1–A8	A9–A14
256K x 1	(512K Bytes)	A1–A9	A10–A18
64K x 4	(128K Bytes)	A1–A8	A9–A16
1M x 1	(2M Bytes)	A1–A10	A11–A19 (+ Bank)
256K x 4	(512K Bytes)	A1–A9	A10–A18

80C188 DESIGNS

NOTE:
Address bit A0 can be used in either $\overline{RAS}$ or $\overline{CAS}$ addresses, so long as it is not included in any refresh address bits.

		Row Address (A0–AX)	Column Address (A0–AX)
64K x 1	(64K Bytes)	A1–A7, A0	A8–A15
16K x 4	(16K Bytes)	A1–A7, A0	A8–A13
256K x 1	(256K Bytes)	A1–A8, A0	A9–A17
64K x 4	(64K Bytes)	A1–A8	A0, A9–A15
1M x 1	(1M Bytes)	A1–A9, A0	A10–A19
256K x 4	(256K Bytes)	A1–A9	A0, A10–A17

RAM Type	$\overline{RAS}$ Add	$\overline{CAS}$ Add	Refresh Add
64K x 1	A0–A7	A0–A7	A0–A6
16K x 4	A0–A7	A0–A5	A0–A6
256K x 1	A0–A8	A0–A8	A0–A7
64K x 4	A0–A7	A0–A7	A0–A7
1M x 1	A0–A9	A0–A9	A0–A8
256K x 4	A0–A8	A0–A8	A0–A8

APPENDIX K
DRAM ADDRESSING CONFIGURATIONS FOR THE 80C186/80C188

80C186 DESIGNS

		Row Address (A0–AX)	Column Address (A0–AX)
64K x 1	(128K Bytes)	A1–A8	A9–A16
16K x 4	(32K Bytes)	A1–A8	A9–A14
256K x 1	(512K Bytes)	A1–A9	A10–A18
64K x 4	(128K Bytes)	A1–A8	A9–A16
1M x 1	(2M Bytes)	A1–A10	A11–A19 (+ Bank)
256K x 4	(512K Bytes)	A1–A9	A10–A18

80C188 DESIGNS

NOTE:
Address bit A0 can be used in either $\overline{RAS}$ or $\overline{CAS}$ address, so long as it is not included in any refresh address bits.

		Row Address (A0–AX)	Column Address (A0–AX)
64K x 1	(64K Bytes)	A1–A7, A0	A8–A15
16K x 4	(16K Bytes)	A1–A7, A0	A8–A13
256K x 1	(256K Bytes)	A1–A8, A0	A9–A17
64K x 4	(64K Bytes)	A1–A8	A0, A9–A16
1M x 1	(1M Bytes)	A1–A9, A0	A10–A19
256K x 4	(256K Bytes)	A1–A9	A0, A10–A17

RAM Type	$\overline{RAS}$ Add	$\overline{CAS}$ Add	Refresh Add
64K x 1	A0–A7	A0–A7	A0–A8
16K x 4	A0–A7	A0–A5	A0–A6
256K x 1	A0–A8	A0–A8	A0–A7
64K x 4	A0–A7	A0–A7	A0–A7
1M x 1	A0–A9	A0–A9	A0–A9
256K x 4	A0–A8	A0–A8	A0–A8

Data Sheets

80186/188/C186/C188
Data Sheets

80186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - — Enhanced 8086-2 CPU
  - — Clock Generator
  - — 2 Independent DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- **Available in 10 MHz (80186-10) and 8 MHz (80186) Versions**
- **High-Performance Processor**
  - — At 8 MHz provides 2 times the Performance of the Standard 8086
  - — 4 MByte/Sec Bus Bandwidth Interface @ 8 MHz
  - — 5 MByte/Sec Bus Bandwidth Interface @ 10 MHz
- **Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086, 8088 Software**
  - — 10 New Instruction Types
- **Complete System Development Support**
  - — Development Software: ASM 86 Assembler, PL/M-86, Pascal-86, Fortran-86, C-86, and System Utilities
  - — In-Circuit-Emulator (I²ICE™-186)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**
- **Available in 68 Pin:**
  - — Plastic Leaded Chip Carrier (PLCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Ceramic Leadless Chip Carrier (LCC)

  (See Packaging Outlines and Dimensions, Order #231369)
- **Available in EXPRESS**
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)

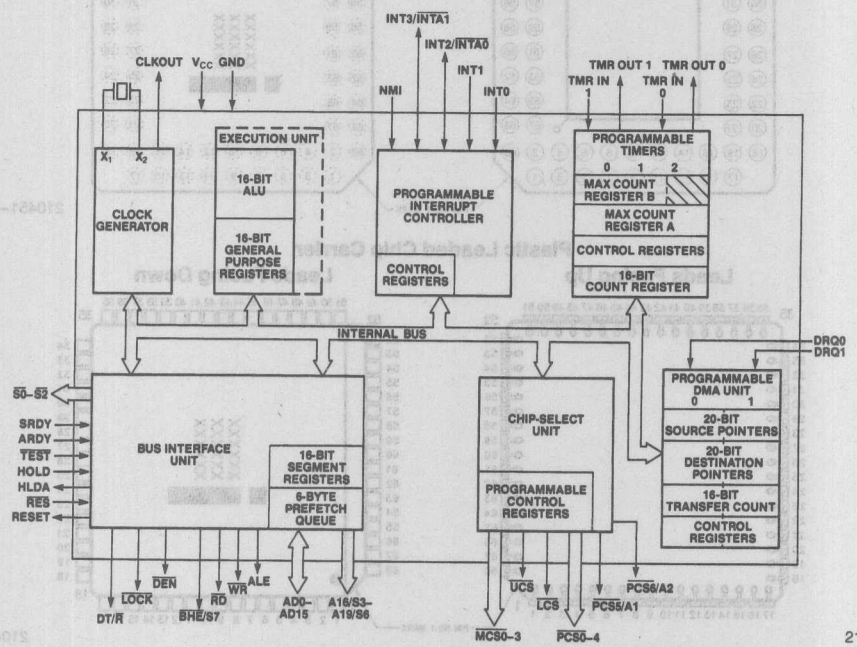


Figure 1. 80186 Block Diagram

October 1988
Order Number: 210451-010

The Intel 80186 is a highly integrated 16-bit microprocessor. The 80186 effectively combines 15–20 of the most common 8086 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz 8086. The 80186 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

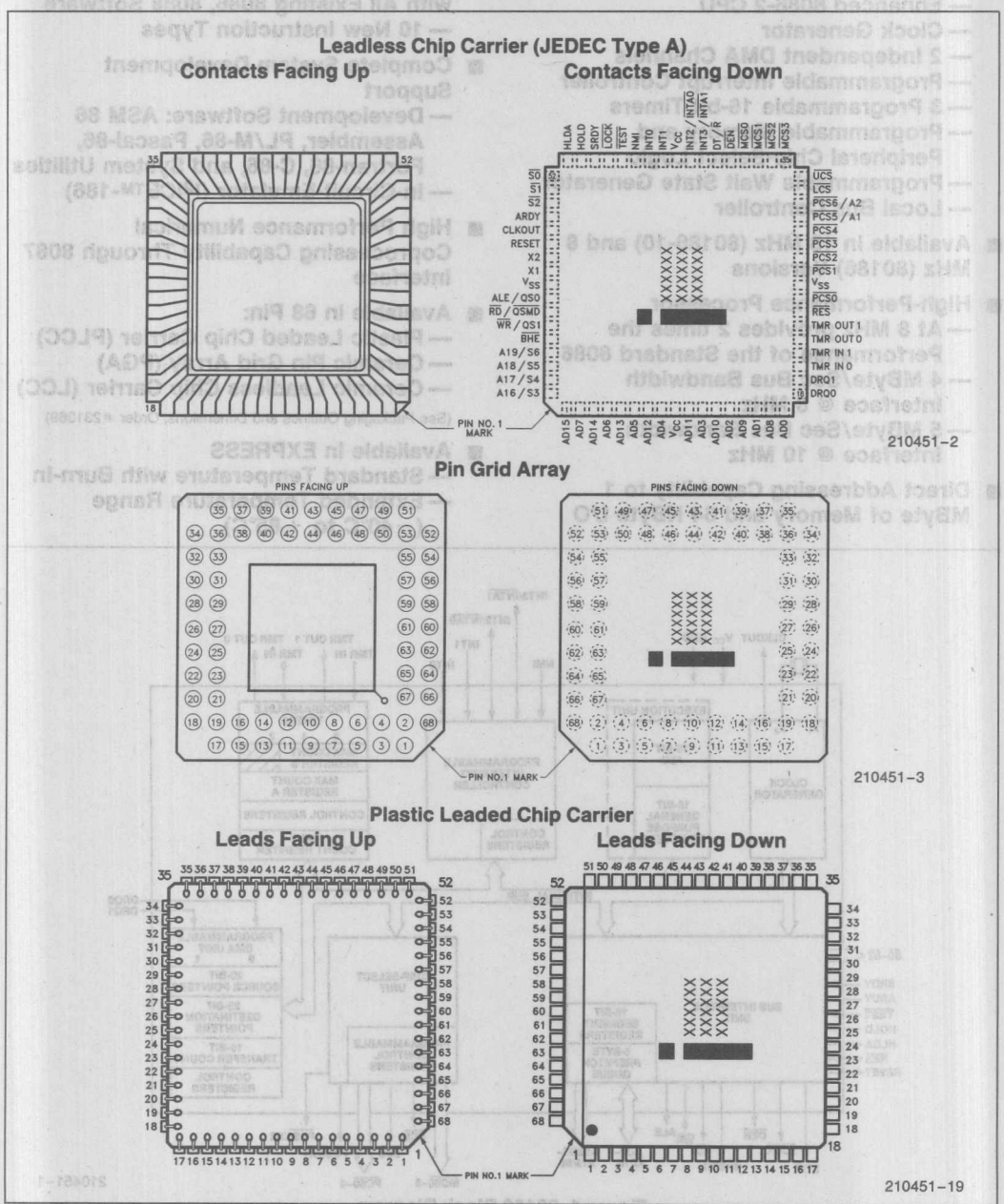


Figure 2. 80186 Pinout Diagrams

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	9 43	I	**System Power:** +5 volt power supply.
V_{SS}	26 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1 X2	59 58	I O	Crystal Inputs X1 and X2 provide external connections for a fundamental mode parallel resonant crystal for the internal oscillator. Instead of using a crystal, an external clock may be applied to X1 while minimizing stray capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT.
$\overline{RES}$	24	I	An active $\overline{RES}$ causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 6½ clock cycles after $\overline{RES}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{RES}$ held LOW. $\overline{RES}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network.
$\overline{TEST}$	47	I	$\overline{TEST}$ is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0 TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0 TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is asserted HIGH by an external device when it is ready for DMA Channel 0 or 1 to perform a transfer. These signals are level-triggered and internally synchronized.
NMI	46	I	The Non-Maskable Interrupt input causes a Type 2 interrupt. An NMI transition from LOW to HIGH is latched and synchronized internally, and initiates the interrupt at the next instruction boundary. NMI must be asserted for at least one clock. The Non-Maskable Interrupt cannot be avoided by programming.
INT0 INT1 INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45 44 42 41	I I I/O I/O	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
A19/S6 A18/S5 A17/S4 A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) indicate the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: **S6: Low = Processor Cycle; High = DMA Cycle** S3, S4, and S5 are defined as LOW during T_2–T_4. The status pins float during HOLD/HLDA.
AD15 AD14 AD13 AD12 AD11 AD10 AD9 AD8 AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0	1 3 5 7 10 12 14 16 2 4 6 8 11 13 15 17	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.
$\overline{BHE}$/S7	64	O	During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus; pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. $\overline{BHE}$/S7 floats during HOLD. **$\overline{BHE}$ and A0 Encodings** **$\overline{BHE}$ Value / A0 Value / Function** 0 / 0 — Word Transfer 0 / 1 — Byte Transfer on upper half of data bus (D15–D8) 1 / 0 — Byte Transfer on lower half of data bus (D_7–D_0) 1 / 1 — Reserved
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during HOLD. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction. **QS1 / QS0 / Queue Operation** 0 / 0 — No queue operation 0 / 1 — First opcode byte fetched from the queue 1 / 1 — Subsequent byte fetched from the queue 1 / 0 — Empty the queue

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}/\overline{QSMD}$	62	I/O	Read Strobe is an active LOW signal which indicates that the 80186 is performing a memory or I/O read cycle. It is guaranteed not to go LOW before the A/D bus is floated. An internal pull-up ensures that $\overline{RD}$ is HIGH during RESET. Following RESET the pin is sampled to determine whether the 80186 is to provide ALE, $\overline{RD}$, and $\overline{WR}$, or queue status information. To enable Queue Status Mode, $\overline{RD}$ must be connected to GND. $\overline{RD}$ will float during bus hold.
ARDY	55	I	Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT, and is active HIGH. The falling edge of ARDY must be synchronized to the 80186 clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.
SRDY	49	I	Synchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows a relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchronize the ARDY input signal. Connecting SRDY high will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the ARDY pin.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of that instruction. No instruction prefetching will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is driven HIGH for one clock during RESET and then floated.
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	52 53 54	O O O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information: **80186 Bus Cycle Status Information** $\overline{S2}$ $\overline{S1}$ $\overline{S0}$ — Bus Cycle Initiated 0 0 0 — Interrupt Acknowledge 0 0 1 — Read I/O 0 1 0 — Write I/O 0 1 1 — Halt 1 0 0 — Instruction Fetch 1 0 1 — Read Data from Memory 1 1 0 — Write Data to Memory 1 1 1 — Passive (no bus cycle) The status pins float during HOLD/HLDA. $\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD HLDA	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$ $\overline{MCS1}$ $\overline{MCS2}$ $\overline{MCS3}$	38 37 36 35	O O O O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$ $\overline{PCS2}$ $\overline{PCS3}$ $\overline{PCS4}$	25 27 28 29 30	O O O O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as a data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80186. The 80186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8086. The 80186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the 8086/8088 instruction set.

80186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes.

Register Set

The 80186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used for arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

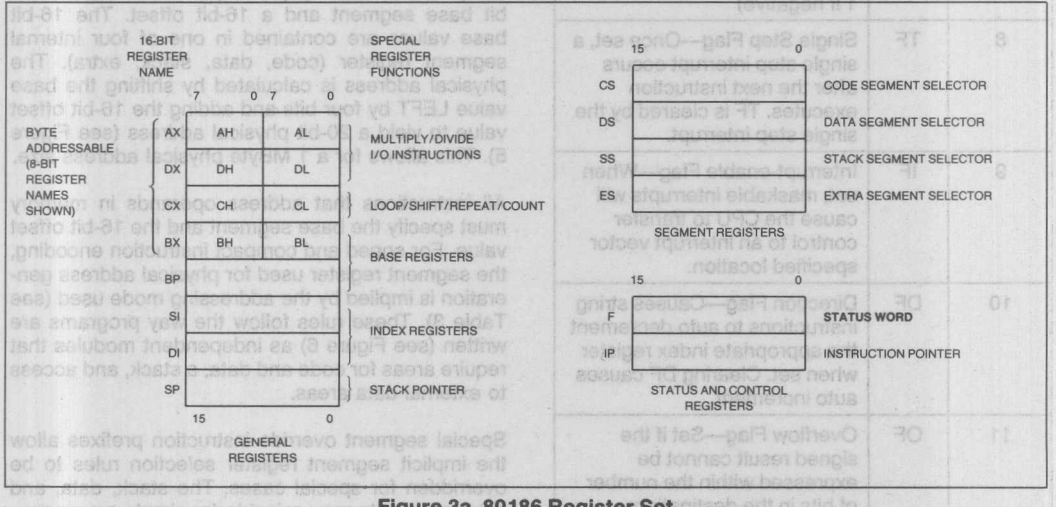


Figure 3a. 80186 Register Set

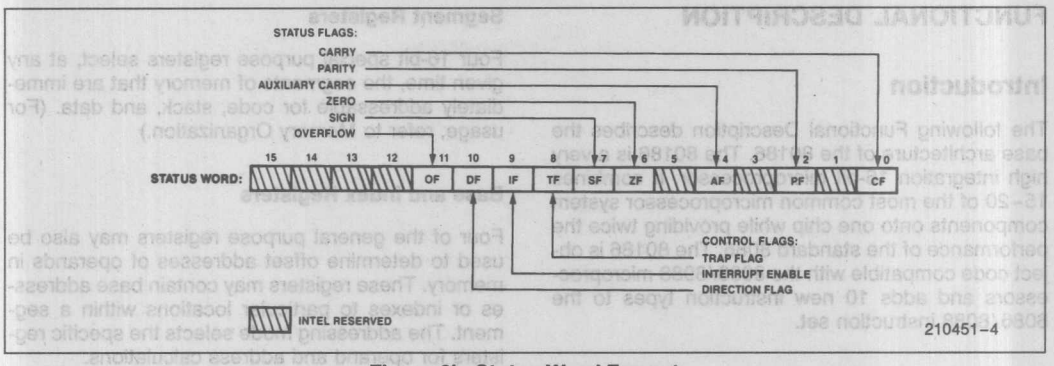


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80186 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

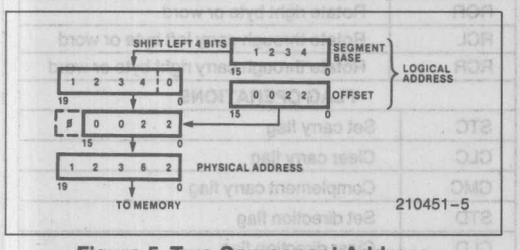


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

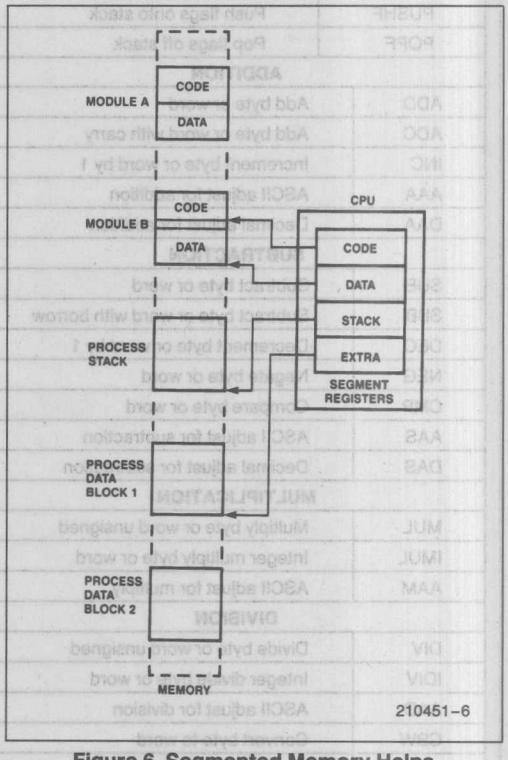


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using an 8087 Numeric Data Coprocessor with the 80186.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using an 8087 Numeric Data Coprocessor with the 80186.)

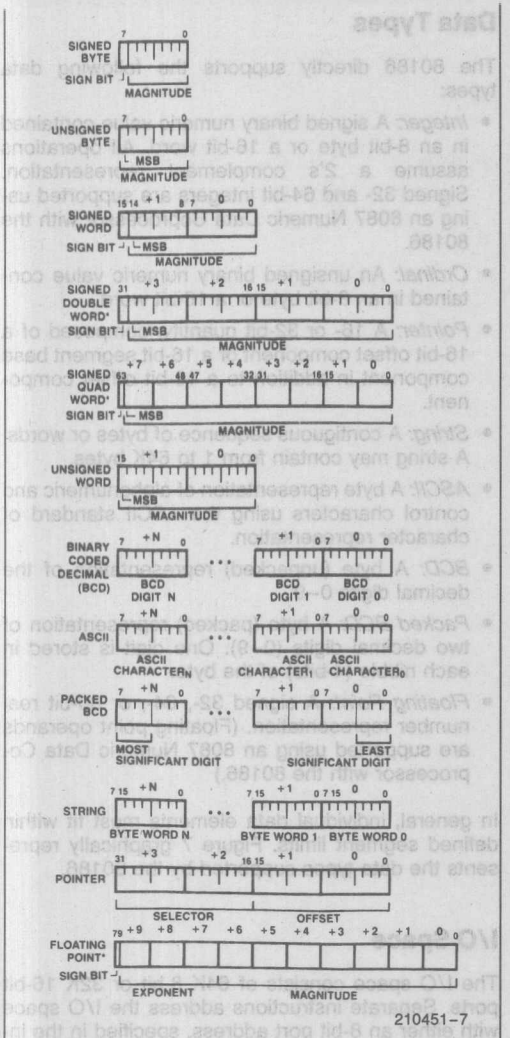
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported by using an 8087 Numeric Data Coprocessor with the 80186.

Figure 7. 80186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80186 Interrupt Vectors

Interrupt Name	Vector Type	Vector Address	Default Priority	Related Instructions	Applicable Notes
Divide Error Exception	0	00H	1	DIV, IDIV	1
Single Step Interrupt	1	04H	1A	All	2
Non-Maskable Interrupt (NMI)	2	08H	1	All	
Breakpoint Interrupt	3	0CH	1	INT	1
INT0 Detected Overflow Exception	4	10H	1	INT0	1
Array Bounds Exception	5	14H	1	BOUND	1
Unused Opcode Exception	6	18H	1	Undefined Opcodes	1
ESC Opcode Exception	7	1CH	1	ESC Opcodes (Coprocessor)	1, 3
Timer 0 Interrupt	8	20H	2A		4
Timer 1 Interrupt	18	48H	2B		4
Timer 2 Interrupt	19	4CH	2C		4
Reserved	9	24H	3		
DMA 0 Interrupt	10	28H	4		
DMA 1 Interrupt	11	2CH	5		
INT0 Interrupt	12	30H	6		
INT1 Interrupt	13	34H	7		
INT2 Interrupt	14	38H	8		
INT3 Interrupt	15	3CH	9		
Reserved	16, 17	40H, 44H			
Reserved	20–31	50H–7CH			

NOTES:
Default priorities for the interrupt sources are used only if the user does not program each source to a unique priority level.
1. Generated as a result of an instruction execution.
2. Performed in same manner as 8086.
3. An ESC (coprocessor) opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.
4. All three timers constitute one source of request to the interrupt controller. As such, they share the same priority level with respect to other interrupt sources. However, the timers have a defined priority order among themselves (2A > 2B > 2C).

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80186

integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80186 Initial Register State after RESET

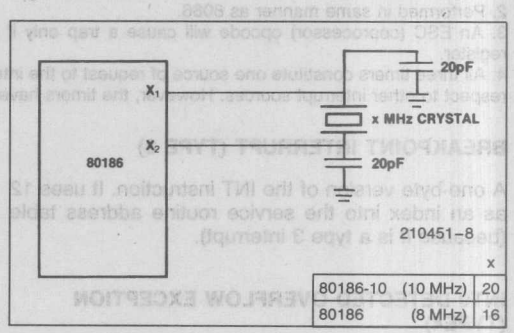
Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80186 CLOCK GENERATOR

The 80186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to the input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80186. The recommended crystal configuration is shown in Figure 8.



		x
80186-10	(10 MHz)	20
80186	(8 MHz)	16

Figure 8. Recommended 80186 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mW max

Clock Generator

The 80186 clock generator provides the 50% duty cycle processor clock for the 80186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3, and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used for full synchronization of a rising ARDY signal. A high-to-low transition on ARDY may be used as an indication of the not ready condition but it must be performed synchronously to CLKOUT **either** in the middle of T_2, T_3, or T_W, **or** at the falling edge of T_3 or T_W.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated. This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80186 provides both a $\overline{RES}$ input pin and a synchronized RESET output pin for use with other system components. The $\overline{RES}$ input pin on the 80186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to approximately two and one-half clocks behind $\overline{RES}$.

Multiple 80186 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides outputs that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80186 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory or I/O to the 80186 or to strobe data from the 80186 to memory or I/O. The ALE line provides a strobe to latch the address when it is valid. The 80186 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80186 generates two control signals to be connected to transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. External circuitry must arbitrate which external device will gain control of the bus when there is more than one alternate local bus master. When the 80186 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{BHE}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

During RESET the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the inactive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–15, A16–19, $\overline{BHE}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80186 integrated peripherals are controlled by 16-bit registers contained within an internal 256-byte control block. The control block may be mapped into either memory or I/O space. Internal logic will recognize control block addresses and respond to bus cycles. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80186 CPU at any time.

The control block base address is programmed by a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space. If the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register

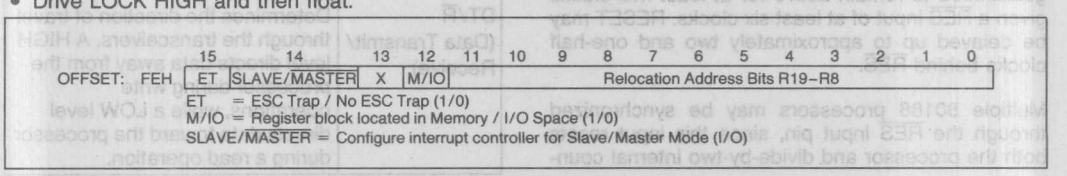


Figure 9. Relocation Register

is set to 20FFH, which maps the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

CHIP-SELECT/READY GENERATION LOGIC

The 80186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

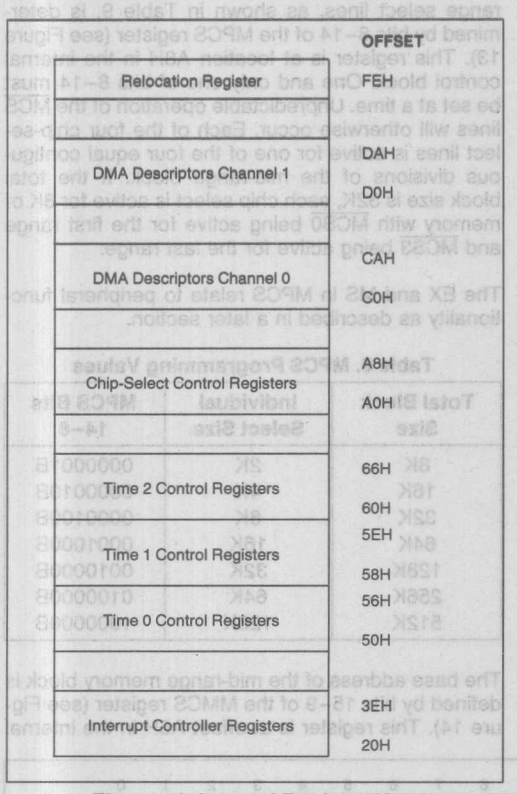


Figure 10. Internal Register Map

Memory Chip Selects

The 80186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{CS}$

The 80186 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

The internal generation of any 20-bit address whose upper 16 bits are equal to or greater than the UMCS

value (with bits 0–5 as "0") asserts $\overline{UCS}$. UMCS bits R2–R0 specify the ready mode for the area of memory defined by this chip select register, as explained later.

Lower Memory $\overline{CS}$

The 80186 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12) at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will assert $\overline{LCS}$. LMCS register bits R2–R0 specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80186 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located within the 80186 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. If the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal

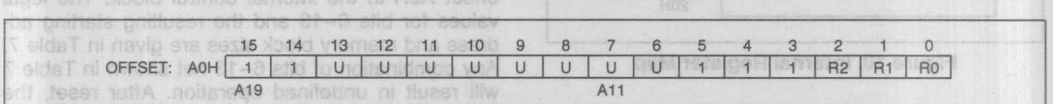


Figure 11. UMCS Register

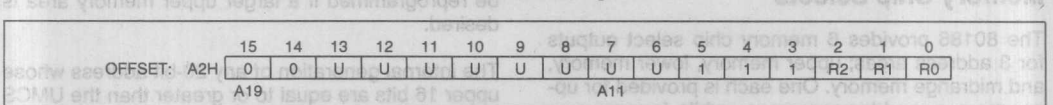


Figure 12. LMCS Register

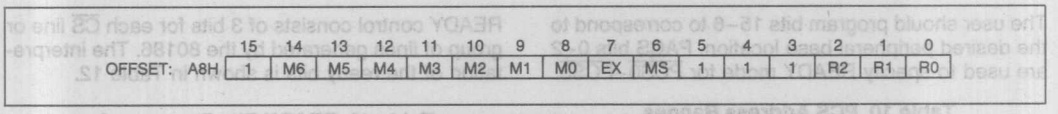


Figure 13. MPCS Register

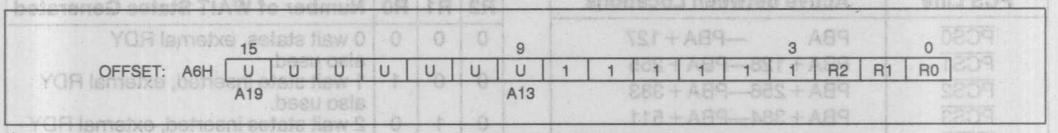


Figure 14. MMCS Register

control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

MMCS bits R2–R0 specify READY mode of operation for all four mid-range chip selects.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The 80186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. The base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1 and A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0 and A1 pins used for selecting internal registers of 8-bit peripheral chips. This simplifies the external hardware because the peripheral registers can be located on even boundaries in I/O or memory space.

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). The register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

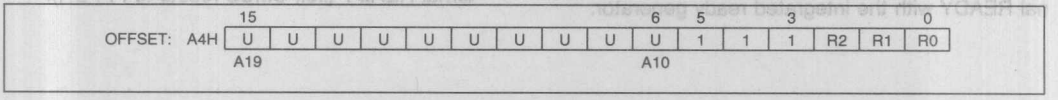


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

$\overline{PCS}$ Line	Active between Locations
$\overline{PCS0}$	PBA —PBA + 127
$\overline{PCS1}$	PBA + 128—PBA + 255
$\overline{PCS2}$	PBA + 256—PBA + 383
$\overline{PCS3}$	PBA + 384—PBA + 511
$\overline{PCS4}$	PBA + 512—PBA + 639
$\overline{PCS5}$	PBA + 640—PBA + 767
$\overline{PCS6}$	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). The register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 specify the READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80186 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). For example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon RESET, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to insert 3 wait states in conjunction with external READY (i.e., UMCS resets to FFFBH).

- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

The 80186 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of 1.25 Mword/sec or 2.5 MBytes/sec at 10 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Count Register, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

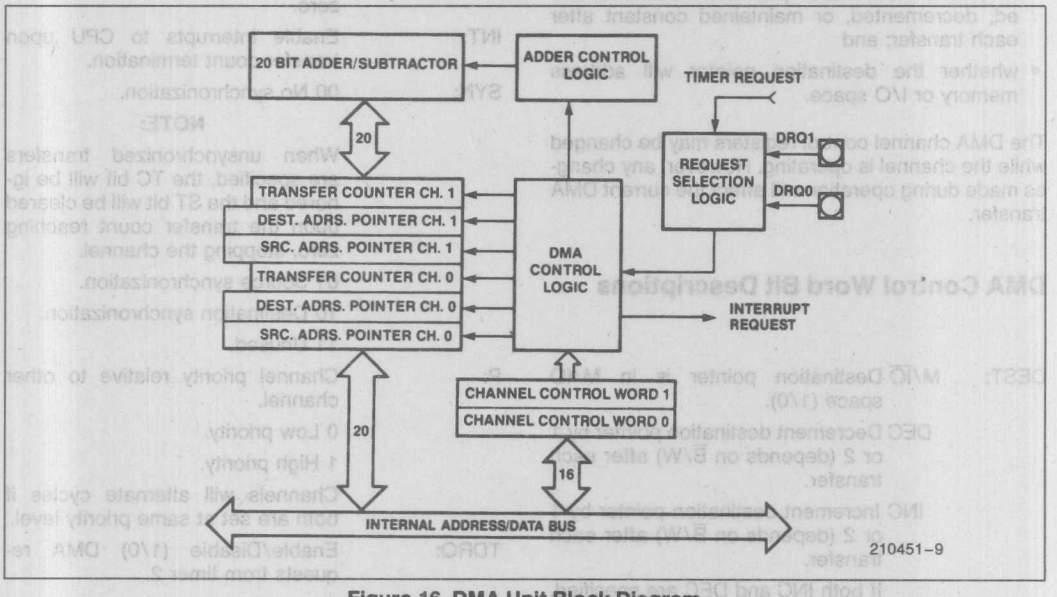


Figure 16. DMA Unit Block Diagram

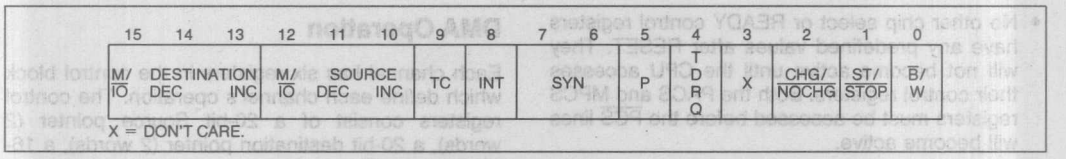


Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 81086 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

DEST: M/$\overline{IO}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

INC Increment destination pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

If both INC and DEC are specified, the pointer will not change after each cycle.

SOURCE: M/$\overline{IO}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

INC Increment source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

If both INC and DEC are specified, the pointer will remain constant after each cycle.

TC: If set, DMA will terminate when the contents of the transfer count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point. If cleared, the DMA controller will decrement the transfer count register for each DMA cycle, but DMA transfers will not stop when the transfer count register reaches zero.

INT: Enable interrupts to CPU upon transfer count termination.

SYN: 00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

P: Channel priority relative to other channel.

0 Low priority.

1 High priority.

Channels will alternate cycles if both are set at same priority level.

TDRQ: Enable/Disable (1/0) DMA requests from timer 2.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be read as 0.

ST/$\overline{\text{STOP}}$: Start/Stop (1/0) channel.

$\overline{\text{B}}$/W: Byte/Word (0/1) transfers.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single bus cycle.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). The register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). When source synchronized or unsynchronized transfers are performed, data will not be fetched from the source address until the destination device signals that it is ready to receive it. Also, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another destination synchronized DMA cycle will begin after two processor clocks. This allows the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. Table 14 shows the maximum DMA transfer rates.

Table 14. Maximum DMA Transfer Rates @ 10 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2.5MBytes/sec	2.5MBytes/sec
Source Synch.	2.5MBytes/sec	2.5MBytes/sec
Destination Synch.	1.7MBytes/sec	2.0MBytes/sec

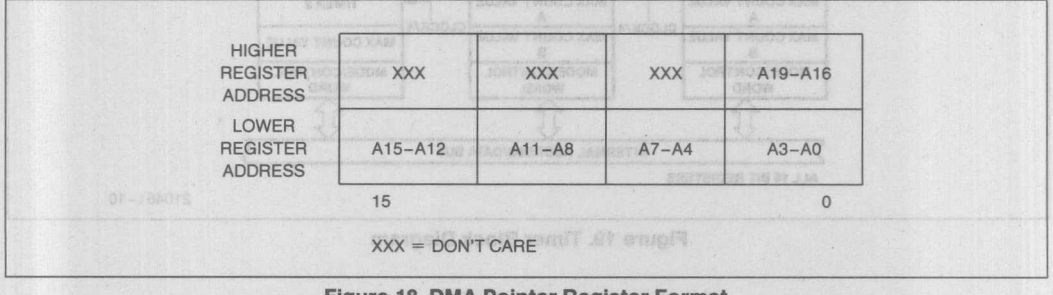


Figure 18. DMA Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed to give one channel priority over the other, or they may be programmed to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; also, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also be generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before the ST/$\overline{\text{STOP}}$ bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/$\overline{\text{STOP}}$ bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, the third timer can be used as a prescaler to the other two, or as a DMA request source.

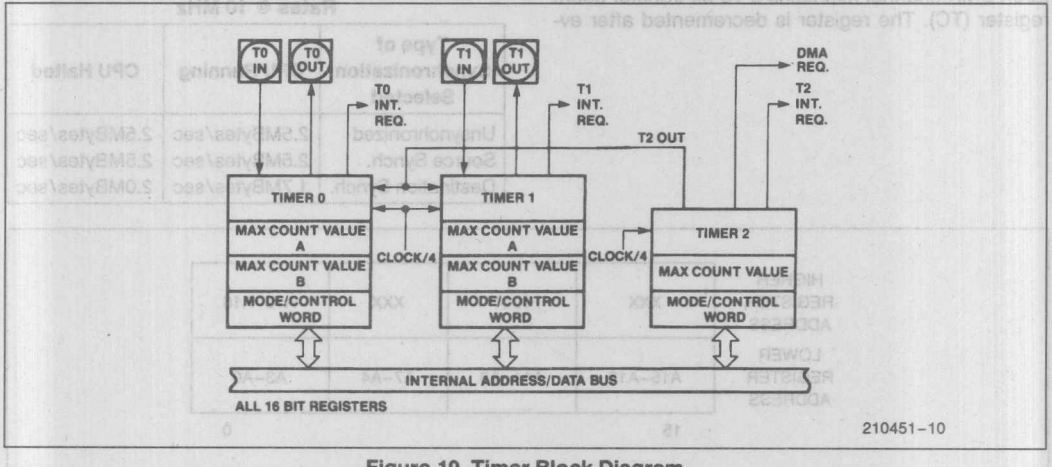


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

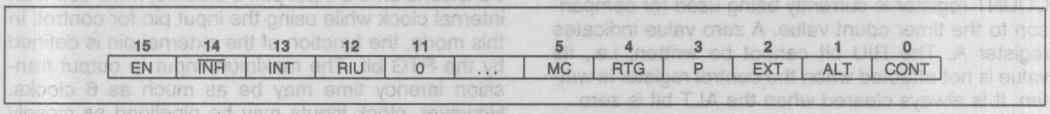


Figure 20. Timer Mode/Control Register

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation.This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse

outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The contents of this register may be read or written by the processor at any time. If the register is written while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. A timer resets when the timer count register equals the max count value being used. If the timer count register or the max count register is changed so that the max count is less than the timer count, the timer does not immediately reset. Instead, the timer counts up to 0FFFFH, "wraps around" to zero, counts up to the max count value, and then resets.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- For Timers 0 and 1, the RIU bits are reset to zero and the ALT bits are set to one. This results in the Timer Out pins going high.

INTERRUPT CONTROLLER

The 80186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80186 interrupt controller has its own control register that sets the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service rountines for lower priority interrupts may be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80186 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

Five pins are provided for external interrupt sources. One of these pins is NMI, the non-maskable interrupt. NMI is generally used for unusual events such as power-fail interrupts. The other four pins may be configured in any of the following ways:

- As four interrupt input lines with internally generated interrupt vectors.
- As an interrupt line and interrupt acknowledge line pair (cascade mode) with externally generated interrupt vectors plus two interrupt input lines with internally generated vectors.
- As two pairs of interrupt/interrupt acknowledge lines (cascade mode) with externally generated interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two INTA cycles are initiated and the vector is read into the 80186 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled, yet be suspended only by interrupts of higher priority than the in-service interrupt.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is executed at the end of the service routine just before the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

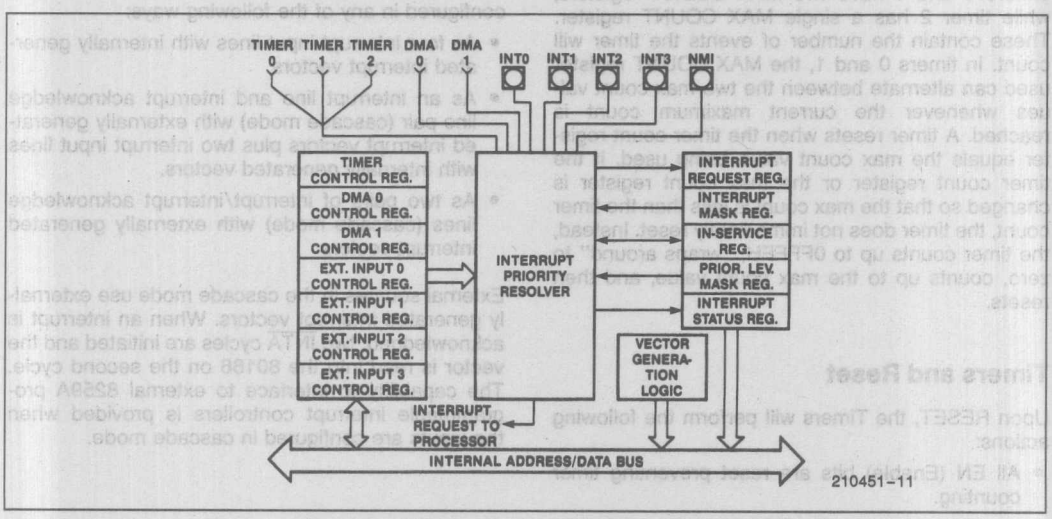


Figure 21. Interrupt Controller Block Diagram

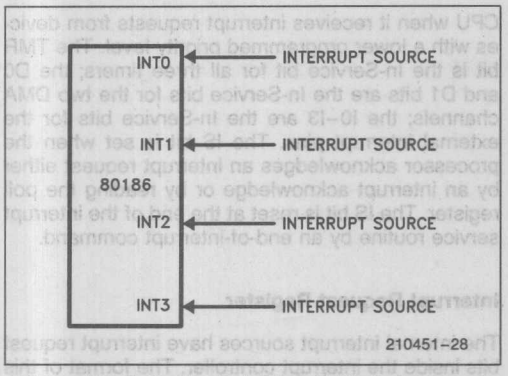


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80186 controller until the 80186 in-service bit is reset. In special fully nested mode, the 80186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the IS register in the external master 8259A is required to determine if there is more than one bit set. If so, the IS bit in the 80186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, other interrupt requests can be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger

mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits are set when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

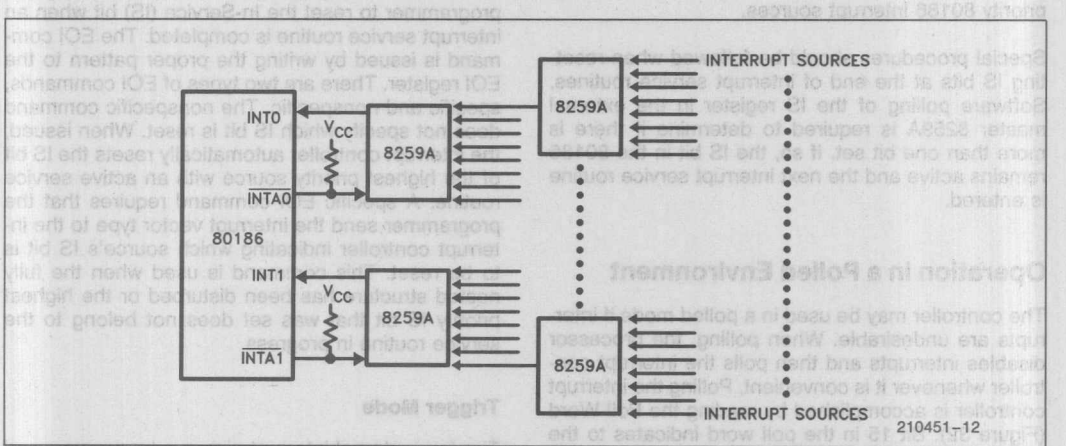


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corresponding to a particular source masks the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

Priority Mask Register

This register masks all interrupts below a particular interrupt priority level. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. This bit allows prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits differentiate between timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14	•	•	•	10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14	•	•	•	•	•	•	•	•	•	3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14	•	•	•	•	•	7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80186 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

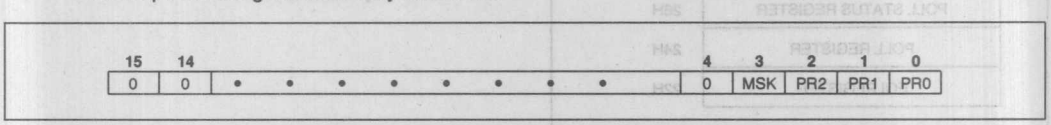


Figure 28. Timer/DMA Control Registers Formats

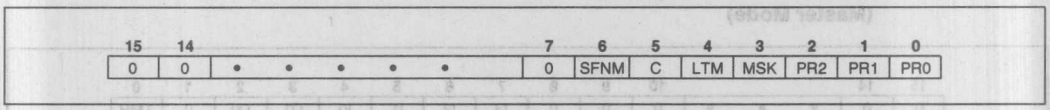


Figure 29. INT0/INT1 Control Register Formats

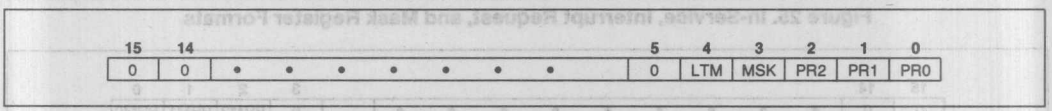


Figure 30. INT2/INT3 Control Register Formats

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80186 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

These level assignments must remain fixed in the iRMX 86 mode of operation.

Slave Mode External Interface

The configuration of the 80186 with respect to an external 8259A master is shown in Figure 33. The INT0 (pin 45) input is used as the 80186 CPU interrupt input. INT3 (pin 41) functions as an output to send the 80186 slave-interrupt-request to one of the 8 master-PIC-inputs.

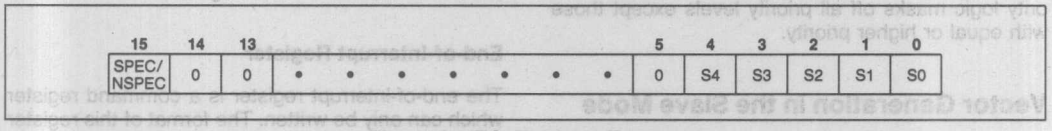


Figure 31. EOI Register Format

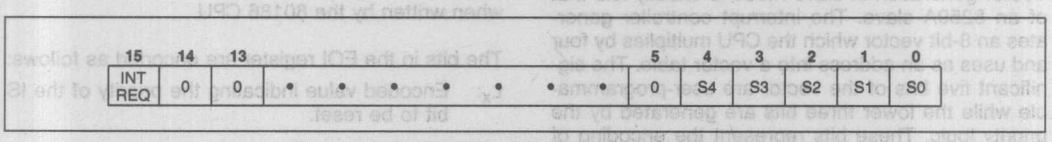


Figure 32. Poll and Poll Status Register Format

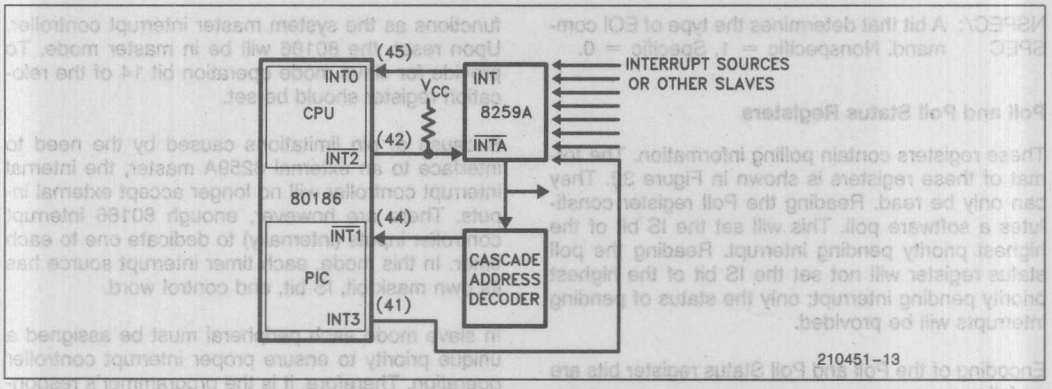


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 8259As do this internally. Because of pin limitations, the 80186 slave address will have to be decoded externally. $\overline{INT1}$ (pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ (pin 42) is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented. (See Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

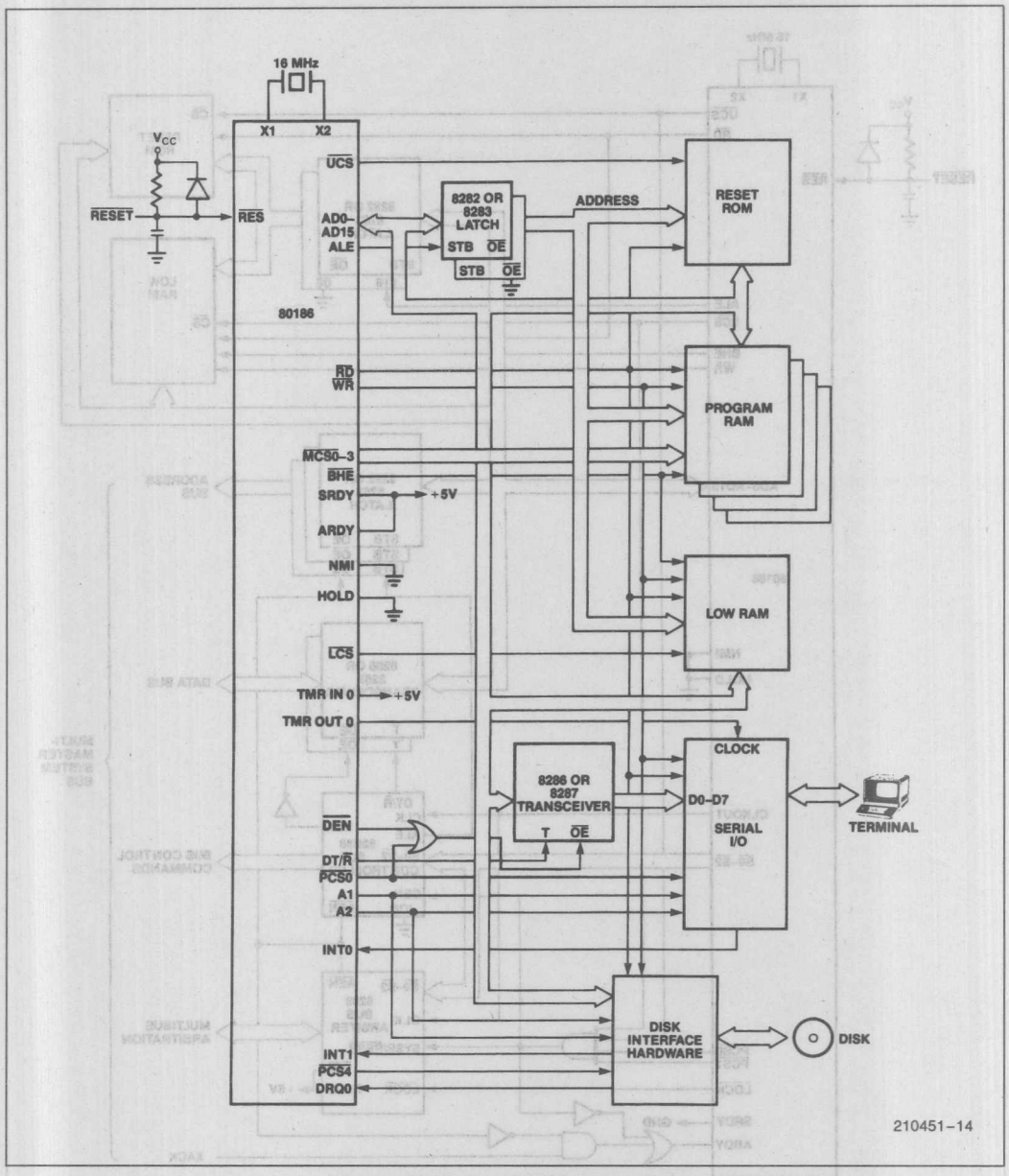


Figure 40. Typical 80186 Computer

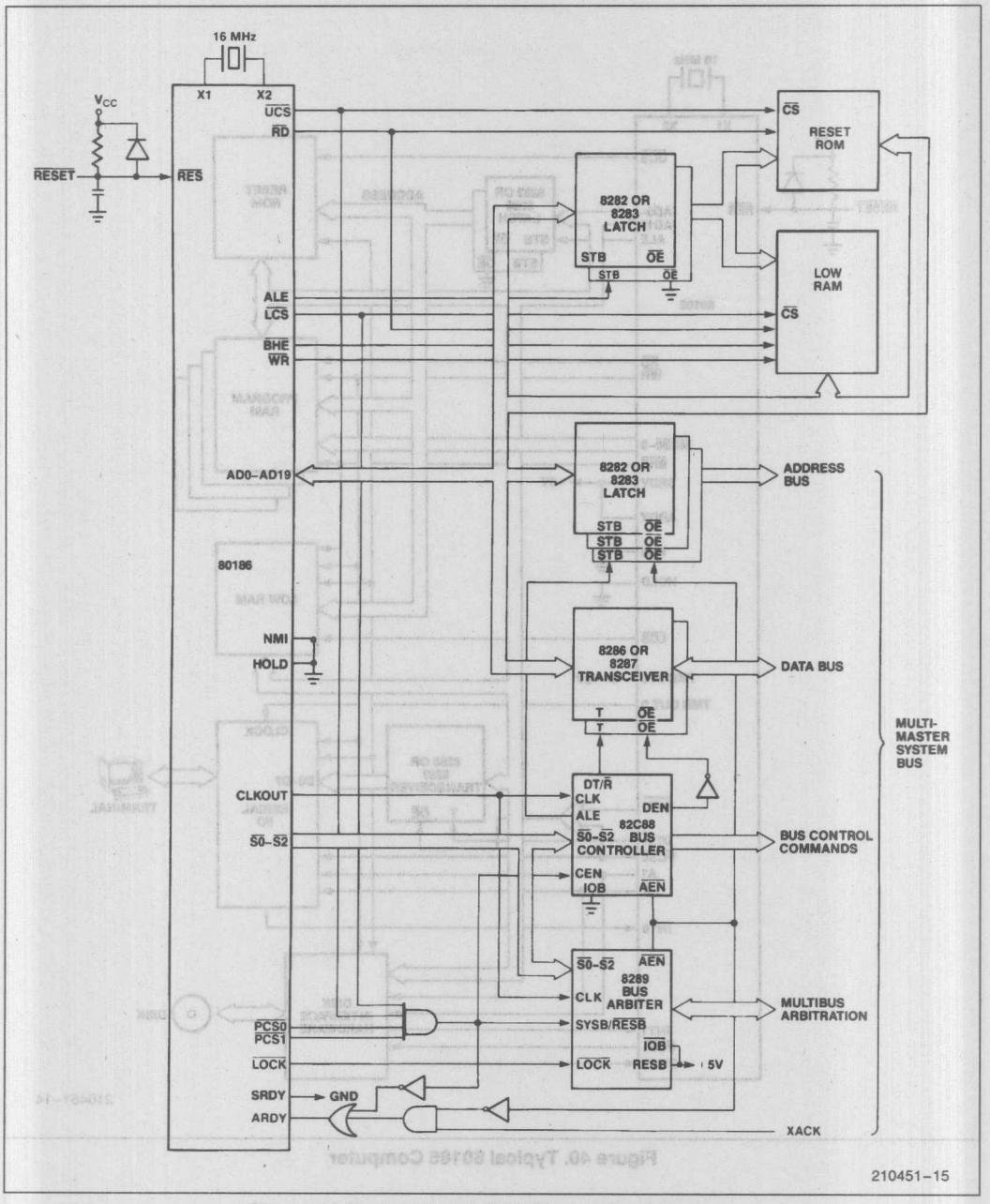


Figure 41. Typical 80186 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on any Pin with Respect to Ground.....−1.0V to +7V
Power Dissipation 3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

Applicable to 80186 (8 MHz), 80186-10 (10 MHz).

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other Outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

80186 Timing Requirements All Timings Measured At 1.5V Unless Otherwise Noted.

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		ns	
T_{ARYHCH}	Asynchronous Ready (ARDY) Active Setup Time (1)	20		15		ns	
T_{ARYLCL}	ARDY Inactive Setup Time	35		25		ns	
T_{CLARX}	ARDY Hold Time	15		15		ns	
T_{ARYCHL}	Asynchronous Ready Inactive Hold Time	15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time (2)	20		20		ns	
T_{CLSRY}	SRDY Transition Hold Time (2)	15		15		ns	
T_{HVCL}	HOLD Setup (1)	25		20		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIM IN, Setup (1)	25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup (1)	25		20		ns	
80186 Master Interface Timing Responses							
T_{CLAV}	Address Valid Delay	5	55	5	44	ns	C_L = 20–200 pF all Outputs (Except T_{CLTMV}) @ 8 & 10 MHz
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	ns	
T_{CHCZ}	Command Lines Float Delay		45		40	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		55		45	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30	ns	
T_{CHLL}	ALE Inactive Delay		35		30	ns	
T_{LLAX}	Address Hold from ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	5	40	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	ns	

1. To guarantee recognition at next clock.
2. To guarantee proper operation.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%) (Continued)

80186 Master Interface Timing Responses (Continued)

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		ns	
T_{AVLL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	ns	
T_{CLTMV}	Timer Output Delay		60		48	ns	100 pF max @ 8 & 10 MHz
T_{CLRO}	Reset Delay		60		48	ns	
T_{CHQSV}	Queue Status Delay		35		28	ns	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to Clock High	10		10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	ns	
80186 Chip-Select Timing Responses							
T_{CLCSV}	Chip-Select Active Delay		66		45	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	ns	
80186 CLKIN Requirements							
T_{CKIN}	CLKIN Period	62.5	250	50	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		ns	1.5V
80186 CLKOUT Timing (200 pF load)							
T_{CICO}	CLKIN to CLKOUT Skew		50		25	ns	
T_{CLCL}	CLKOUT Period	125	500	100	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12	ns	3.5 to 1.0V

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has from 5 to 7 characters. The first character is always a "T" (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

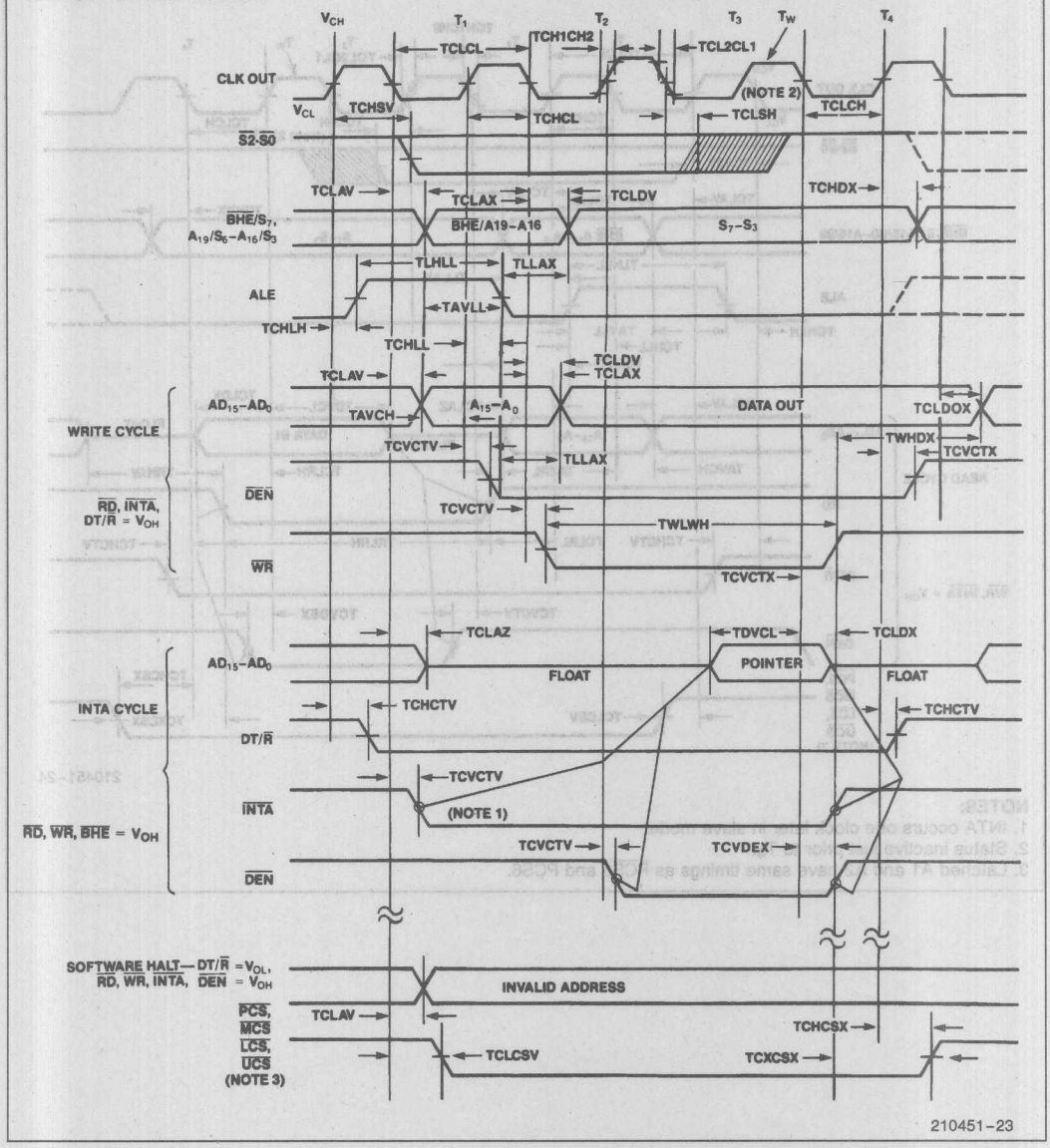
A: Address
ARY: Asynchronous Ready Input
C: Clock Output
CK: Clock Input
CS: Chip Select
CT: Control (DT/$\overline{R}$, $\overline{DEN}$, . . .)
D: Data Input
DE: $\overline{DEN}$
H: Logic Level High
IN: Input (DRQ0, TIM0, . . .)
L: Logic Level Low or ALE
O: Output
QS: Queue Status (QS1, QS2)
R: $\overline{RD}$ signal, RESET signal
S: Status ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$)
SRY: Synchronous Ready Input
V: Valid
W: WR Signal
X: No Longer a Valid Logic Level
Z: Float

Examples:

T_{CLAV} — Time from Clock low to Address valid
T_{CHLH} — Time from Clock high to ALE high
T_{CLCSV} — Time from Clock low to Chip Select valid

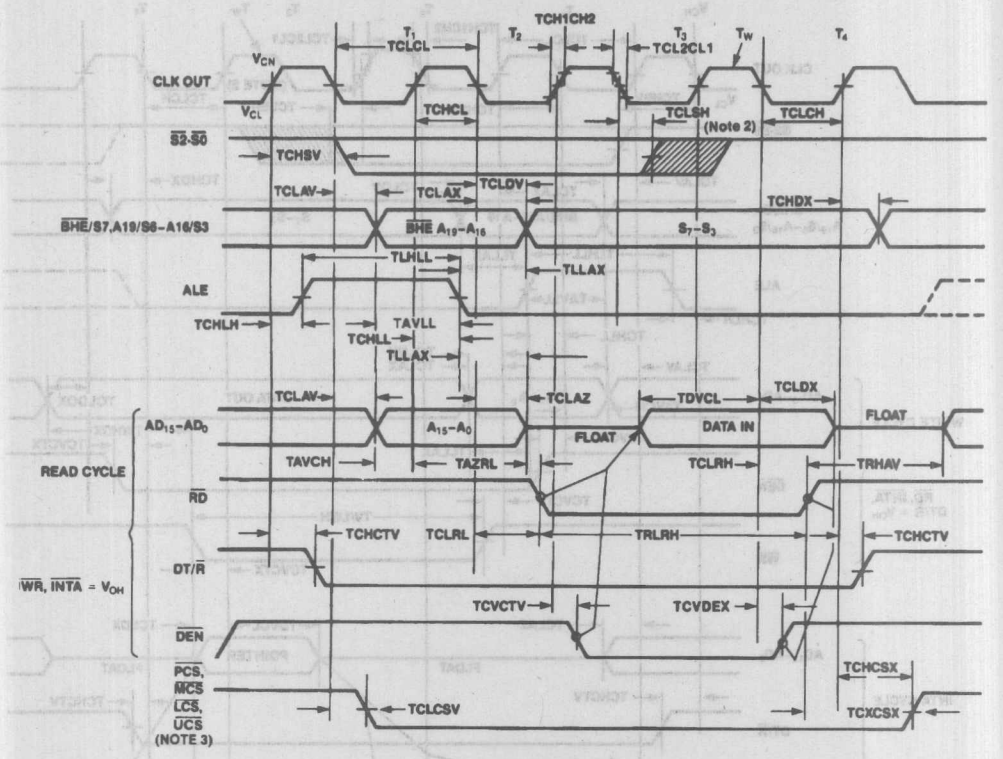
WAVEFORMS

MAJOR CYCLE TIMING



WAVEFORMS (Continued)

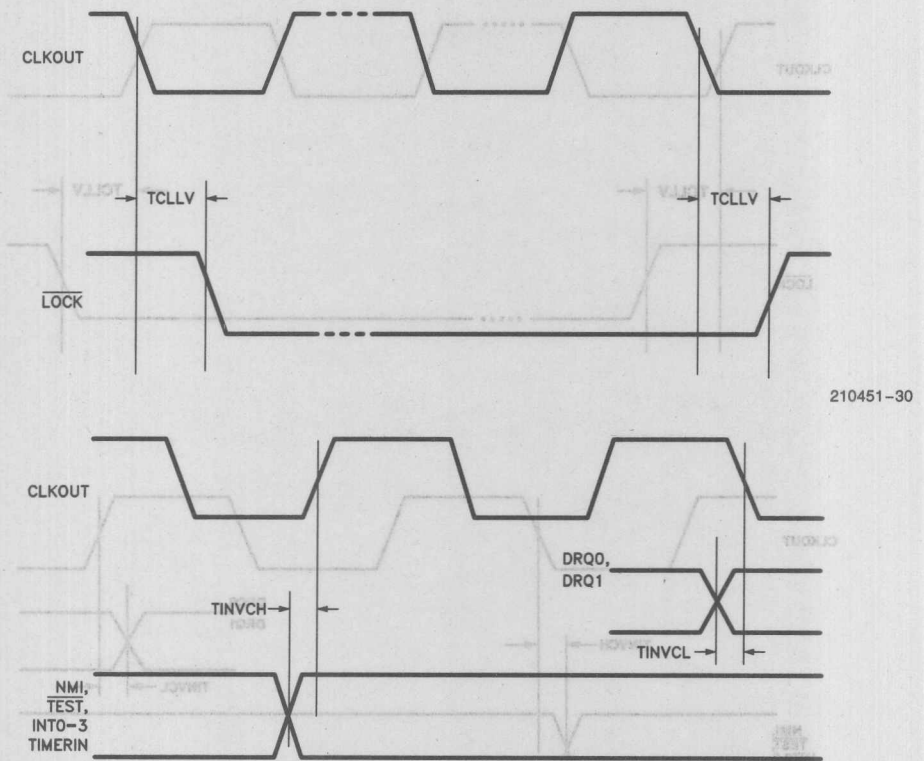
MAJOR CYCLE TIMING (Continued)



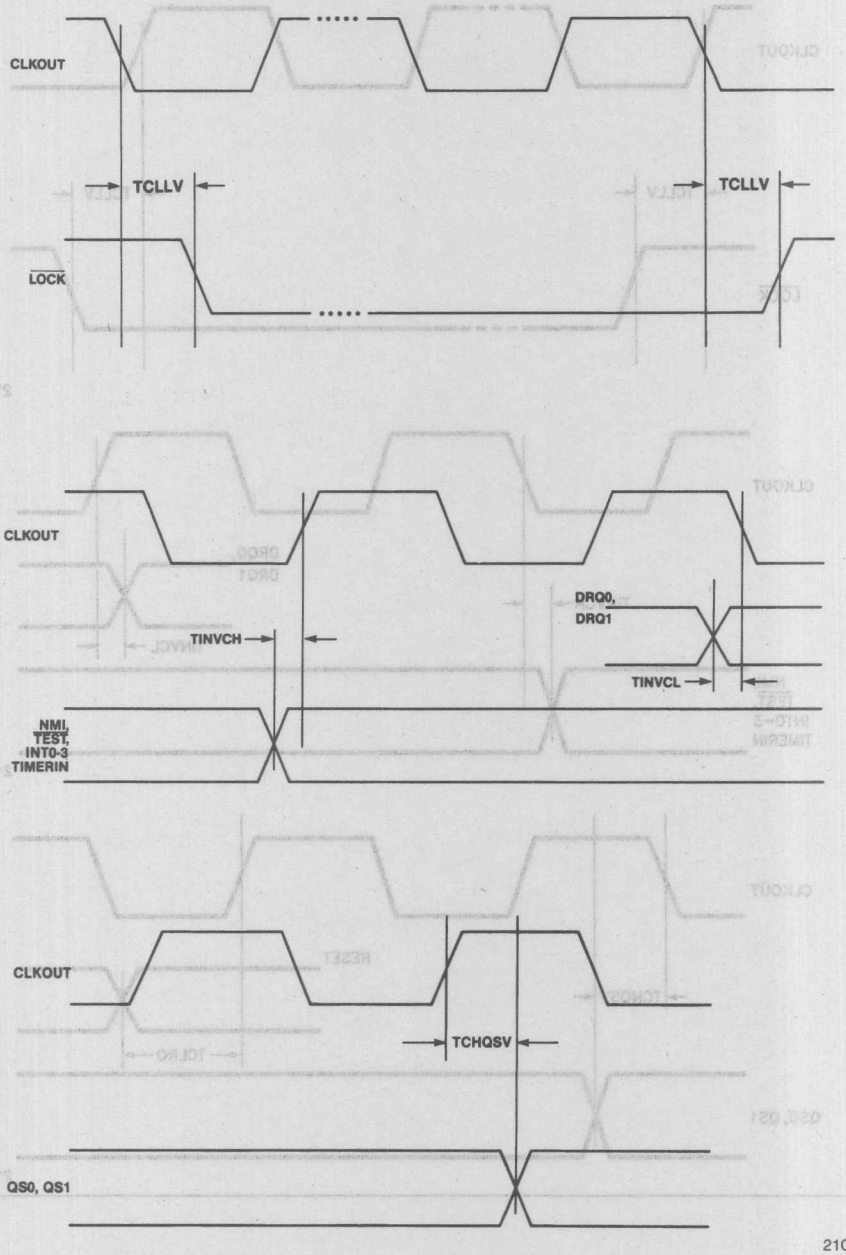
210451-24

NOTES:
1. INTA occurs one clock later in slave mode.
2. Status inactive just prior to T_4.
3. Latched A1 and A2 have same timings as $\overline{PCS5}$ and $\overline{PCS6}$.

WAVEFORMS (Continued)

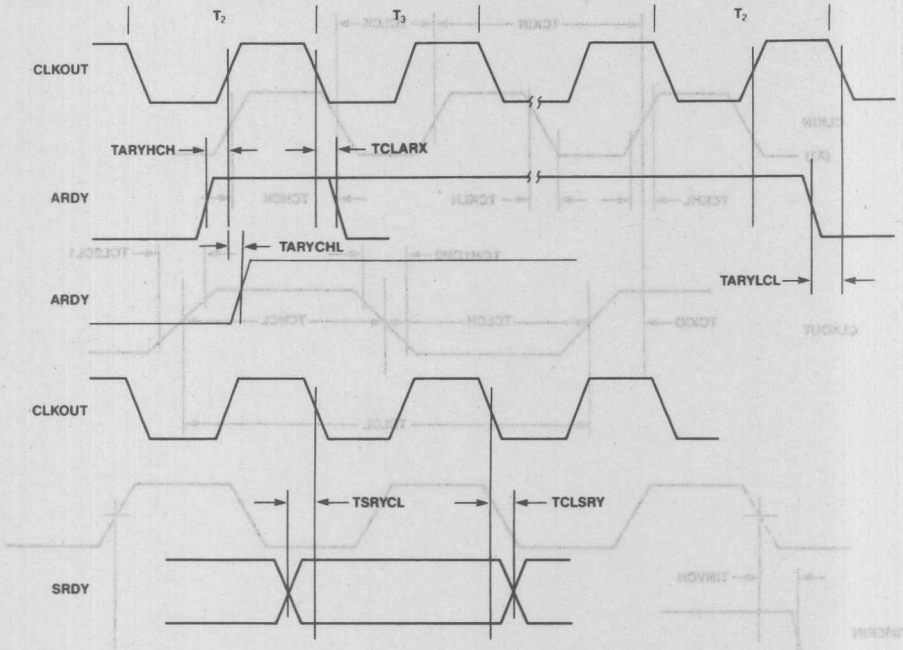


WAVEFORMS (Continued)



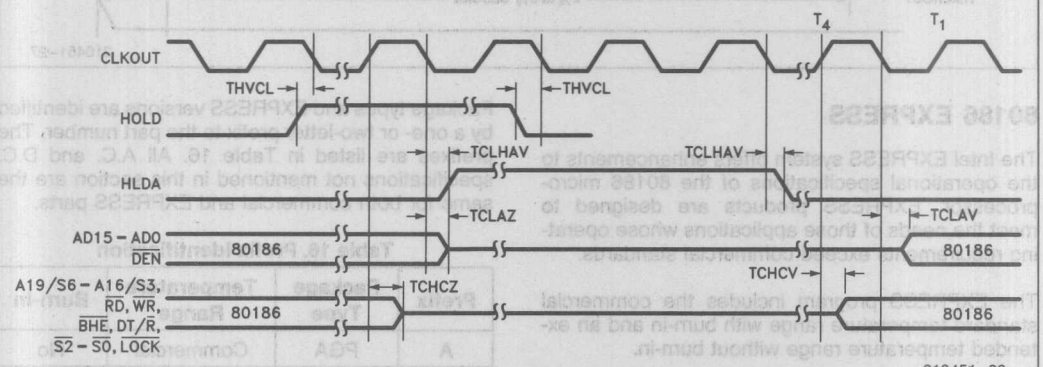
WAVEFORMS (Continued)

READY TIMING



210451-29

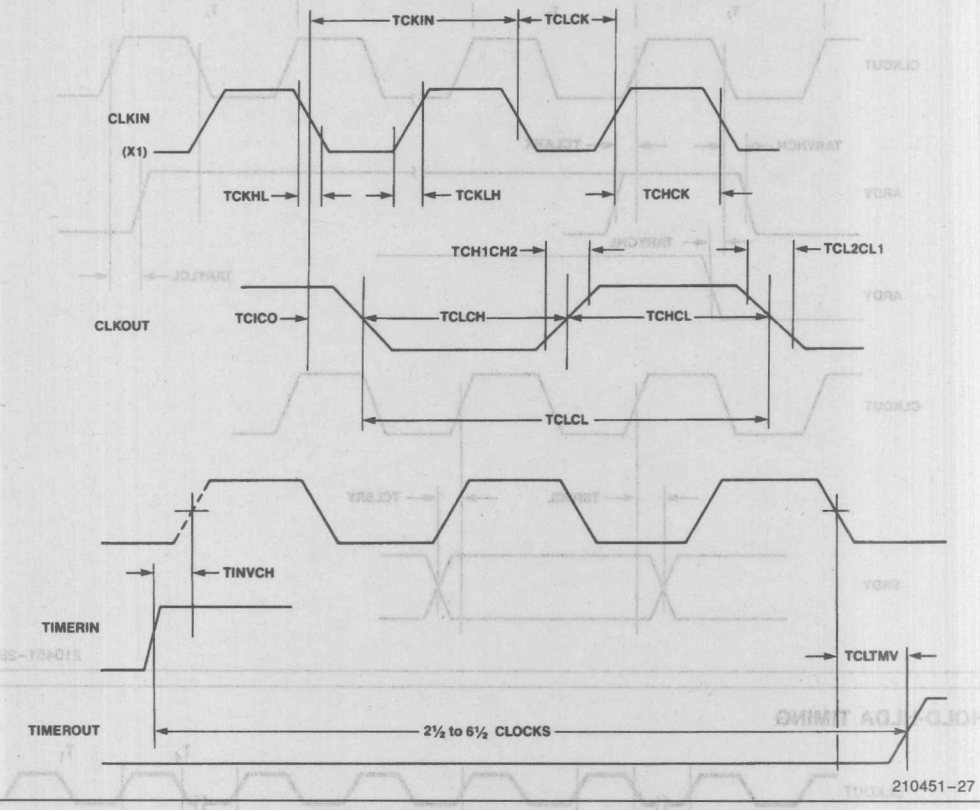
HOLD-HLDA TIMING



210451-26

WAVEFORMS (Continued)

TIMER ON 80186



80186 EXPRESS

The Intel EXPRESS system offers enhancements to the operational specifications of the 80186 microprocessor. EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in and an extended temperature range without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to +70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at +125°C with V_{CC} = 5.5V ±0.25V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 16. All A.C. and D.C. specifications not mentioned in this section are the same for both commercial and EXPRESS parts.

Table 16. Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
A	PGA	Commercial	No
N	PLCC	Commercial	No
R	LCC	Commercial	No
TA	PGA	Extended	No
QA	PGA	Commercial	Yes
QR	LCC	Commercial	Yes

NOTE:
Not all package/temperature range/speed combinations are available.

80186 EXECUTION TIMINGS

Since the bus interface unit and execution unit operate independently, a determination of 80186 program execution timing must consider the bus cycles necessary to prefetch instructions as well as the number of EU cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All instructions which involve memory accesses can also require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

With a 16-bit BIU, the 80186 has sufficient bus performance to ensure that an adequate number of prefetched bytes will reside in the queue most of the time. Therefore, actual program execution time will not be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1000100w	mod reg r/m			2/12	
Register/memory to register	1000101w	mod reg r/m			2/9	
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1	12–13	8/16-bit
Immediate to register	1011w reg	data	data if w=1		3–4	8/16-bit
Memory to accumulator	1010000w	addr-low	addr-high		8	
Accumulator to memory	1010001w	addr-low	addr-high		9	
Register/memory to segment register	10001110	mod 0 reg r/m			2/9	
Segment register to register/memory	10001100	mod 0 reg r/m			2/11	
PUSH = Push:						
Memory	11111111	mod 1 1 0 r/m			16	
Register	01010 reg				10	
Segment register	000reg110				9	
Immediate	011010s0	data	data if s=0		10	
PUSHA = Push All	01100000				36	
POP = Pop:						
Memory	10001111	mod 0 0 0 r/m			20	
Register	01011 reg				10	
Segment register	000reg111	(reg≠01)			8	
POPA = Pop All	01100001				51	
XCHG = Exchange:						
Register/memory with register	1000011w	mod reg r/m			4/17	
Register with accumulator	10010 reg				3	
IN = Input from:						
Fixed port	1110010w	port			10	
Variable port	1110110w				8	
OUT = Output to:						
Fixed port	1110011w	port			9	
Variable port	1110111w				7	
XLAT = Translate byte to AL	11010111				11	
LEA = Load EA to register	10001101	mod reg r/m			6	
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		18	
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠11)		18	
LAHF = Load AH with flags	10011111				2	
SAHF = Store AH into flags	10011110				3	
PUSHF = Push flags	10011100				9	
POPF = Pop flags	10011101				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
DATA TRANSFER (Continued)			
SEGMENT = Segment Override:			
CS	00101110	2	
SS	00110110	2	
DS	00111110	2	
ES	00100110	2	
ARITHMETIC			
ADD = Add:			
Reg/memory with register to either	000000dw | mod reg r/m	3/10	
Immediate to register/memory	100000sw | mod 000 r/m | data | data if s w=01	4/16	
Immediate to accumulator	0000010w | data | data if w=1	3/4	8/16-bit
ADC = Add with carry:			
Reg/memory with register to either	000100dw | mod reg r/m	3/10	
Immediate to register/memory	100000sw | mod 010 r/m | data | data if s w=01	4/16	
Immediate to accumulator	0001010w | data | data if w=1	3/4	8/16-bit
INC = Increment:			
Register/memory	1111111w | mod 000 r/m	3/15	
Register	01000 reg	3	
SUB = Subtract:			
Reg/memory and register to either	001010dw | mod reg r/m	3/10	
Immediate from register/memory	100000sw | mod 101 r/m | data | data if s w=01	4/16	
Immediate from accumulator	0010110w | data | data if w=1	3/4	8/16-bit
SBB = Subtract with borrow:			
Reg/memory and register to either	000110dw | mod reg r/m	3/10	
Immediate from register/memory	100000sw | mod 011 r/m | data | data if s w=01	4/16	
Immediate from accumulator	0001110w | data | data if w=1	3/4	8/16-bit
DEC = Decrement			
Register/memory	1111111w | mod 001 r/m	3/15	
Register	01001 reg	3	
CMP = Compare:			
Register/memory with register	0011101w | mod reg r/m	3/10	
Register with register/memory	0011100w | mod reg r/m	3/10	
Immediate with register/memory	100000sw | mod 111 r/m | data | data if s w=01	3/10	
Immediate with accumulator	0011110w | data | data if w=1	3/4	8/16-bit
NEG = Change sign register/memory	1111011w | mod 011 r/m	3/10	
AAA = ASCII adjust for add	00110111	8	
DAA = Decimal adjust for add	00100111	4	
AAS = ASCII adjust for subtract	00111111	7	
DAS = Decimal adjust for subtract	00101111	4	
MUL = Multiply (unsigned):	1111011w | mod 100 r/m		
Register-Byte		26–28	
Register-Word		35–37	
Memory-Byte		32–34	
Memory-Word		41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
IMUL = Integer multiply (signed):	1111011w	mod 101 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43	
IMUL = Integer Immediate multiply (signed)	011010s1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1111011w	mod 110 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44	
IDIV = Integer divide (signed):	1111011w	mod 111 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67	
AAM = ASCII adjust for multiply	11010100	00001010			19	
AAD = ASCII adjust for divide	11010101	00001010			15	
CBW = Convert byte to word	10011000				2	
CWD = Convert word to double word	10011001				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1101000w	mod TTT r/m			2/15	
Register/Memory by CL	1101001w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1100000w	mod TTT r/m	count		5+n/17+n	
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR					
AND = **And:**						
Reg/memory and register to either	001000dw	mod reg r/m			3/10	
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w=1	4/16	
Immediate to accumulator	0010010w	data	data if w=1		3/4	8/16-bit
TEST = **And function to flags, no result:**						
Register/memory and register	1000010w	mod reg r/m			3/10	
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w=1	4/10	
Immediate data and accumulator	1010100w	data	data if w=1		3/4	8/16-bit
OR = **Or:**						
Reg/memory and register to either	000010dw	mod reg r/m			3/10	
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w=1	4/16	
Immediate to accumulator	0000110w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
LOGIC (Continued)			
XOR = Exclusive or:			
Reg/memory and register to either	001100dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod 110 r/m | data | data if w=1	4/16	
Immediate to accumulator	0011010w | data | data if w=1	3/4	8/16-bit
NOT = Invert register/memory	1111011w | mod 010 r/m	3/10	
STRING MANIPULATION			
MOVS = Move byte/word	1010010w	14	
CMPS = Compare byte/word	1010011w	22	
SCAS = Scan byte/word	1010111w	15	
LODS = Load byte/wd to AL/AX	1010110w	12	
STOS = Store byte/wd from AL/AX	1010101w	10	
INS = Input byte/wd from DX port	0110110w	14	
OUTS = Output byte/wd to DX port	0110111w	14	
Repeated by count in CX (REP/REPE/REPZ/REPNE/REPNZ)			
MOVS = Move string	11110010 | 1010010w	8+8n	
CMPS = Compare string	1111001z | 1010011w	5+22n	
SCAS = Scan string	1111001z | 1010111w	5+15n	
LODS = Load string	11110010 | 1010110w	6+11n	
STOS = Store string	11110010 | 1010101w	6+9n	
INS = Input string	11110010 | 0110110w	8+8n	
OUTS = Output string	11110010 | 0110111w	8+8n	
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	15	
Register/memory indirect within segment	11111111 | mod 010 r/m	13/19	
Direct intersegment	10011010 | segment offset | segment selector	23	
Indirect intersegment	11111111 | mod 011 r/m (mod ≠ 11)	38	
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 100 r/m	11/17	
Direct intersegment	11101010 | segment offset | segment selector	14	
Indirect intersegment	11111111 | mod 101 r/m (mod ≠ 11)	26	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
RET = **Return from CALL:**						
Within segment	11000011				16	
Within seg adding immed to SP	11000010	data-low	data-high		18	
Intersegment	11001011				22	
Intersegment adding immediate to SP	11001010	data-low	data-high		25	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110 000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = **Interrupt:**						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{TEST}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT mod LLL r/m (TTT LLL are opcode to processor extension)	6	
NOP = No Operation	10010000	3	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

REVISION HISTORY

The sections significantly revised since version -009 are:

Pin Description Table	Various descriptions rewritten for clarity.
Interrupt Vector Table	Redrawn for clarity.
A.C. Characteristics	Added reminder that T_{SRYCL} and T_{CLSRY} must be met.
Explanation of the A.C. Symbols	New section.
Major Cycle Timing Waveforms	T_{CLRO} indicated.

The sections significantly revised since version -008 are:

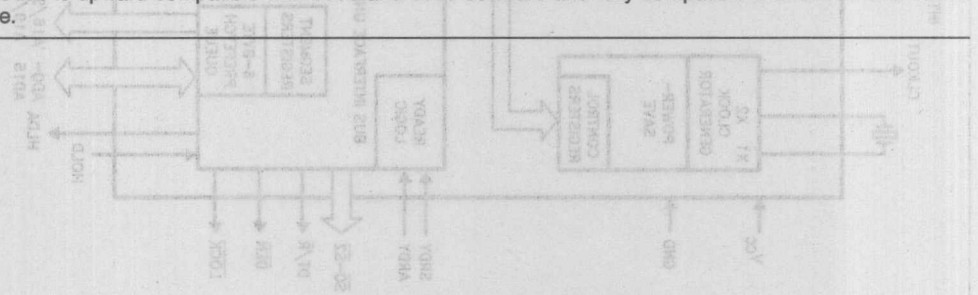
Pin Description Table	Noted $\overline{RES}$ to be low more than 4 clocks. Connections to X1 and X2 clarified.
DMA Control Bit Descriptions	Moved and clarified note concerning TC condition for ST/$\overline{STOP}$ clearing during unsynchronized transfers.
Interrupt Controller, etc.	Renamed iRMX Mode to Slave Mode.
Interrupt Request Register	Noted that D0 and D1 are read/write, others read-only.
Execution Timings	Effect of bus width clarified.

80C186
CHMOS HIGH INTEGRATION 16-BIT MICROPROCESSOR

- Operation Modes Include:
  - Enhanced Mode Which Has
    - DRAM Refresh
    - Power-Save Mode
    - Direct Interface to New Numerics Coprocessor
  - Compatible Mode
    - NMOS 80186 Pin for Pin Replacement for Non-Numerics Applications
- Integrated Feature Set
  - Enhanced 80C86/C88 CPU
  - Clock Generator
  - 2 Independent DMA Channels
  - Programmable Interrupt Controller
  - 3 Programmable 16-Bit Timers
  - Dynamic RAM Refresh Control Unit
  - Programmable Memory and Peripheral Chip Select Logic
  - Programmable Wait State Generator
  - Local Bus Controller
  - Power Save Mode
  - System-Level Testing Support (High Impedance Test Mode)
- Available in 16 MHz (80C186-16), 12.5 MHz (80C186-12) and 10 MHz (80C186) Versions
- Direct Addressing Capability to 1 MByte Memory and 64 KByte I/O
- Completely Object Code Compatible with All Existing 8086/8088 Software and Also Has 10 Additional Instructions over 8086/8088
- Complete System Development Support
  - All 8086 and NMOS 80186 Software Development Tools Can Be Used for 80C186 System Development
    - ASM 86 Assembler, PL/M-86, Pascal-86, Fortran-86, C-86, and System Utilities
    - In-Circuit-Emulator (ICE™-186)
- High Performance Numeric Coprocessing Capability through 80C187 Interface
- Available in 68 Pin:
  - Plastic Leaded Chip Carrier (PLCC)
  - Ceramic Pin Grid Array (PGA)
  - Ceramic Leadless Chip Carrier (JEDEC A Package)

  (See Packaging Outlines and Dimensions, Order Number 231369)
- Available in EXPRESS:
  - Standard Temperature with Burn-In
  - Extended Temperature Range (−40°C to +85°C)

The Intel 80C186 is a CHMOS high integration microprocessor. It has features which are new to the 80186 family which include a DRAM refresh control unit, power-save mode and a direct numerics interface. When used in "compatible" mode, the 80C186 is 100% pin-for-pin compatible with the NMOS 80186 (except for 8087 applications). The "enhanced" mode of operation allows the full feature set of the 80C186 to be used. The 80C186 is upward compatible with 8086 and 8088 software and fully compatible with 80186 and 80188 software.



October 1988
Order Number: 270354-003

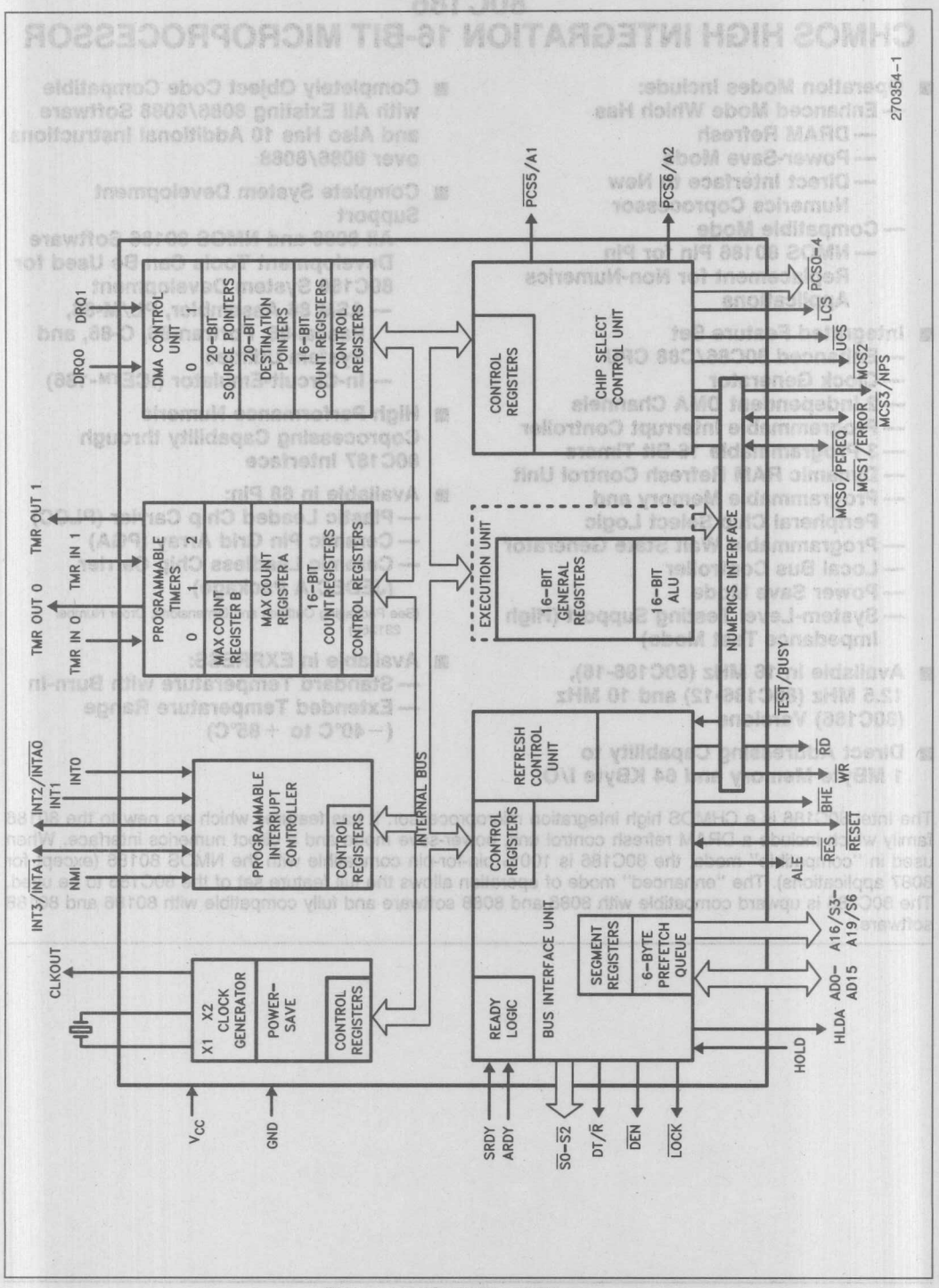


Figure 1. 80C186 Block Diagram

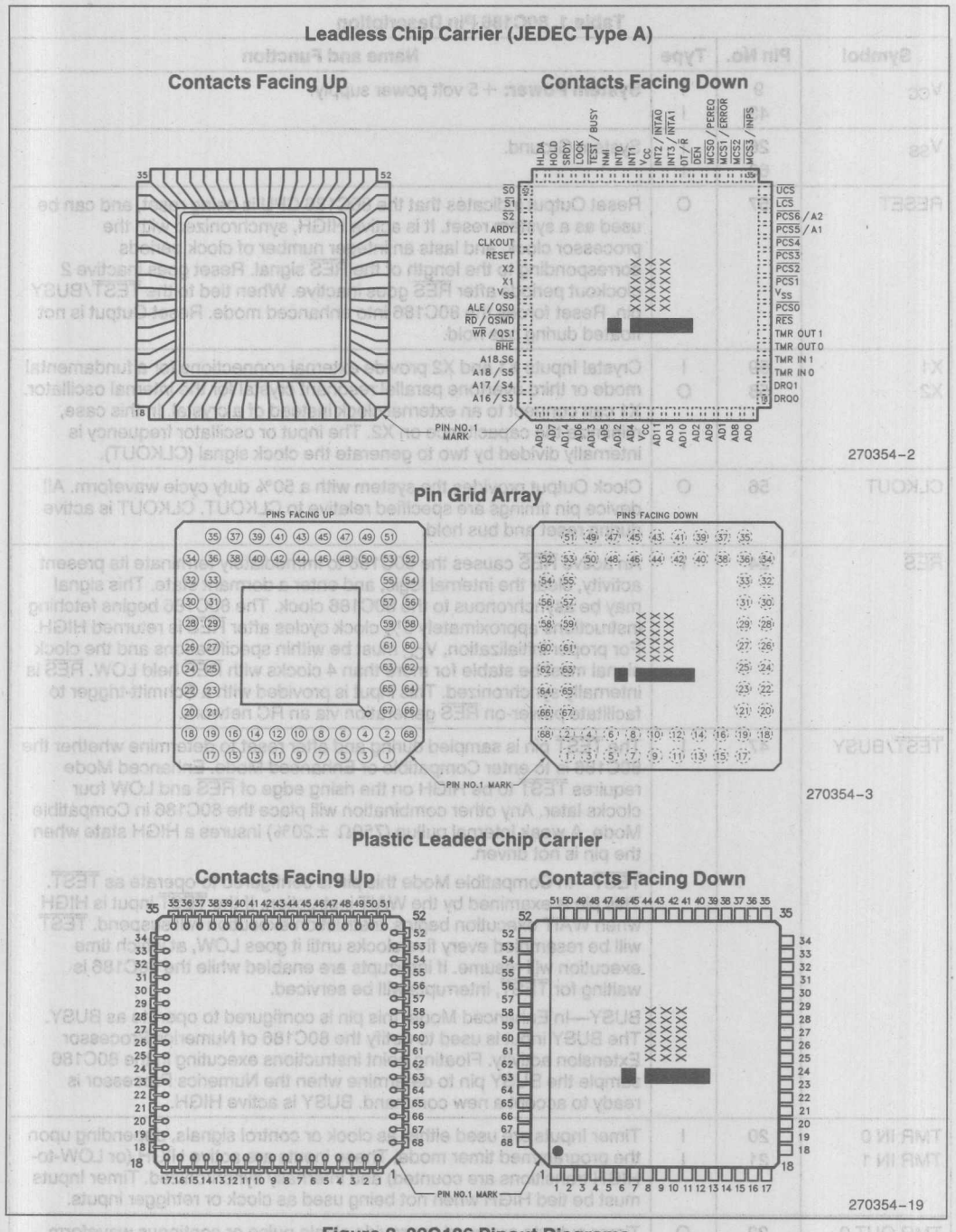


Figure 2. 80C186 Pinout Diagrams

Table 1. 80C186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	9 43	I I	**System Power:** +5 volt power supply.
V_{SS}	26 60	I I	System Ground.
RESET	57	O	Reset Output indicates that the 80C186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal. Reset goes inactive 2 clockout periods after $\overline{RES}$ goes inactive. When tied to the $\overline{TEST}$/BUSY pin, Reset forces the 80C186 into enhanced mode. Reset Output is not floated during bus hold.
X1 X2	59 58	I O	Crystal Inputs X1 and X2 provide external connections for a fundamental mode or third overtone parallel resonant crystal for the internal oscillator. X1 can connect to an external clock instead of a crystal. In this case, minimize the capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT is active during reset and bus hold.
$\overline{RES}$	24	I	An active $\overline{RES}$ causes the 80C186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80C186 clock. The 80C186 begins fetching instructions approximately 6½ clock cycles after $\overline{RES}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{RES}$ held LOW. $\overline{RES}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network.
$\overline{TEST}$/BUSY	47	I	The $\overline{TEST}$ pin is sampled during and after reset to determine whether the 80C186 is to enter Compatible or Enhanced Mode. Enhanced Mode requires $\overline{TEST}$ to be HIGH on the rising edge of $\overline{RES}$ and LOW four clocks later. Any other combination will place the 80C186 in Compatible Mode. A weak internal pullup (750Ω ±20%) insures a HIGH state when the pin is not driven. $\overline{TEST}$—In Compatible Mode this pin is configured to operate as $\overline{TEST}$. This pin is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when WAIT execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled every five clocks until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80C186 is waiting for $\overline{TEST}$, interrupts will be serviced. BUSY—In Enhanced Mode, this pin is configured to operate as BUSY. The BUSY input is used to notify the 80C186 of Numerics Processor Extension activity. Floating point instructions executing in the 80C186 sample the BUSY pin to determine when the Numerics Processor is ready to accept a new command. BUSY is active HIGH.
TMR IN 0 TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized. Timer Inputs must be tied HIGH when not being used as clock or retrigger inputs.
TMR OUT 0 TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected. These outputs are not floated during a bus hold.

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
DRQ0 DRQ1	18 19	I I	DMA Request is asserted HIGH by an external device when it is ready for DMA Channel 0 or 1 to perform a transfer. These signals are level-triggered and internally synchronized.
NMI	46	I	The Non-Maskable Interrupt input causes a Type 2 interrupt. An NMI transition from LOW to HIGH is latched and synchronized internally, and initiates the interrupt at the next instruction boundary. NMI must be asserted for at least one clock. The Non-Maskable Interrupt cannot be avoided by programming.
INT0 INT1 INT2/$\overline{\text{INTA0}}$ INT3/$\overline{\text{INTA1}}$	45 44 42 41	I I I/O I/O	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
A19/S6 A18/S5 A17/S4 A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) indicate the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: **Low** / **High** S6: Processor Cycle / DMA Cycle S3, S4, and S5 are defined as LOW during T_2–T_4. These outputs are floated during bus hold or reset.
AD15 AD14 AD13 AD12 AD11 AD10 AD9 AD8 AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0	1 3 5 7 10 12 14 16 2 4 6 8 11 13 15 17	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{\text{BHE}}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations. These pins are floated during a bus hold or reset.

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{BHE}$	64	O	The $\overline{BHE}$ (Bus High Enable) signal is analogous to A0 in that it is used to enable data on to the most significant half of the data bus, pins D15–D8. $\overline{BHE}$ will be LOW during T_1 when the upper byte is transferred and will remain LOW through T_3 AND T_W. $\overline{BHE}$ does not need to be latched. $\overline{BHE}$ will float during HOLD or reset. In Enhanced Mode, $\overline{BHE}$ will also be used to signify DRAM refresh cycles. A refresh cycle is indicated by $\overline{BHE}$ and A0 being HIGH.
			$\overline{BHE}$ and A0 Encodings
			$\overline{BHE}$ Value / **A0 Value** / **Function** 0 / 0 / Word Transfer 0 / 1 / Byte Transfer on upper half of data bus (D15–D8) 1 / 0 / Byte Transfer on lower half of data bus (D_7–D_0) 1 / 1 / Refresh
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80C186 to latch the address. ALE is active HIGH, with addresses guaranteed valid on the trailing edge. ALE floats during reset, but not during bus hold.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. It is active LOW, and floats during bus hold or reset. When the 80C186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.
			QS1 / **QS0** / **Queue Operation** 0 / 0 / No queue operation 0 / 1 / First opcode byte fetched from the queue 1 / 1 / Subsequent byte fetched from the queue 1 / 0 / Empty the queue
$\overline{RD}$/$\overline{QSMD}$	62	I/O	Read Strobe is an active LOW signal which indicates that the 80C186 is performing a memory or I/O read cycle. It is guaranteed not to go LOW before the A/D bus is floated. An internal pull-up (750Ω ±20%) ensures that $\overline{RD}$/$\overline{QSMD}$ is HIGH during RESET. Following RESET the pin is sampled to determine whether the 80C186 is to provide ALE, $\overline{RD}$, and $\overline{WR}$, or queue status information. To enable Queue Status Mode, $\overline{RD}$ must be connected to GND. $\overline{RD}$ will float during bus hold.
ARDY	55	I	Asynchronous Ready informs the 80C186 that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT and is active HIGH. The falling edge of ARDY must be synchronized to the 80C186 clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.
SRDY	49	I	Synchronous Ready informs the 80C186 that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows a relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchonize the ARDY input signal. Connecting SRDY high will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the ARDY pin.

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus, $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction immediately following the LOCK prefix. It remains active until the completion of that instruction. No instruction prefetching will occur while $\overline{LOCK}$ is asserted. $\overline{LOCK}$ floats during bus hold or reset.
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	52 53 54	O O O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information: **80C186 Bus Cycle Status Information** $\overline{S2}$ $\overline{S1}$ $\overline{S0}$ — Bus Cycle Initiated 0 0 0 — Interrupt Acknowledge 0 0 1 — Read I/O 0 1 0 — Write I/O 0 1 1 — Halt 1 0 0 — Instruction Fetch 1 0 1 — Read Data from Memory 1 1 0 — Write Data to Memory 1 1 1 — Passive (no bus cycle) The status pins float during HOLD/HLDA. $\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
HOLD HLDA	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. The 80C186 generates HLDA (HIGH) in response to a HOLD request. Simultaneous with the issuance of HLDA, the 80C186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80C186 will lower HLDA. When the 80C186 needs to run another bus cycle, it will again drive the local bus and control lines. In Enhanced Mode, HLDA will go low when a DRAM refresh cycle is pending in the 80C186 and an external bus master has control of the bus. It will be up to the external master to relinquish the bus by lowering HOLD so that the 80C186 may execute the refresh cycle. Lowering HOLD for one clock and returning HIGH will ensure only one refresh cycle to the external master. HLDA will immediately go active after the refresh cycle has taken place.
$\overline{UCS}$	34	I/O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. $\overline{UCS}$ does not float during bus hold. The address range activating $\overline{UCS}$ is software programmable. $\overline{UCS}$ and $\overline{LCS}$ are sampled upon the rising edge of $\overline{RES}$. If both pins are held low, the 80C186 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{UCS}$ has a weak internal pullup (750Ω ±20%) that is active during RESET to ensure that the 80C186 does not enter ONCE mode inadvertently.

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{LCS}$	33	I/O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. $\overline{LCS}$ does not float during bus HOLD. The address range activating $\overline{LCS}$ is software programmable. $\overline{UCS}$ and $\overline{LCS}$ are sampled upon the rising edge of $\overline{RES}$. If both pins are held low, the 80C186 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{LCS}$ has a weak internal pullup (750Ω ±20%) that is active only during RESET to ensure that the 80C196 does not enter ONCE mode inadvertently.
$\overline{MCS0}$/PEREQ $\overline{MCS1}$/$\overline{ERROR}$ $\overline{MCS2}$ $\overline{MCS3}$/$\overline{NPS}$	38 37 36 35	O/I O/I O O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines do not float during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable. In Enhanced Mode, $\overline{MCS0}$ becomes a PEREQ input (Processor Extension Request). When connected to the Numerics Processor Extension, this input is used to signal the 80C186 when to make numeric data transfers to and from the NPX. $\overline{MCS3}$ becomes $\overline{NPS}$ (Numeric Processor Select) which may only be activated by communication to the Numerics Processor Extension. $\overline{MCS1}$ becomes $\overline{ERROR}$ in enhanced mode and is used to signal numerics coprocessor errors. $\overline{MCS0}$/PEREQ and $\overline{MSC1}$/$\overline{ERROR}$ have weak internal pullups (750Ω ±20%) which are active during reset.
$\overline{PCS0}$ $\overline{PCS1}$ $\overline{PCS2}$ $\overline{PCS3}$ $\overline{PCS4}$	25 27 28 29 30	O O O O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O or 1 MByte memory space). These lines do not float during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1 rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH. $\overline{PCS5}$/A1 does not float during bus hold.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2 rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH. $\overline{PCS6}$/A2 does not float during bus hold.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the 80C186. When HIGH the 80C186 places write data on the data bus. DT/$\overline{R}$ floats during a bus hold or reset.
$\overline{DEN}$	39	O	Data Enable is provided as a data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state. $\overline{DEN}$ will float during a bus hold or reset.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80C186. The 80C186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip. The 80C186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the 8086/8088 instruction set.

The 80C186 has two major modes of operation, Compatible and Enhanced. In Compatible Mode the 80C186 is completely compatible with NMOS 80186, with the exception of 8087 support. The Enhanced mode adds three new features to the system design. These are Power-Save control, Dynamic RAM refresh, and an asynchronous Numerics Coprocessor interface.

80C186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80188 family all contain the same basic set of registers, instructions, and addressing modes. The 80C186 processor is upward compatible with the 8086 and 8088 CPUs.

Register Set

The 80C186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used for arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

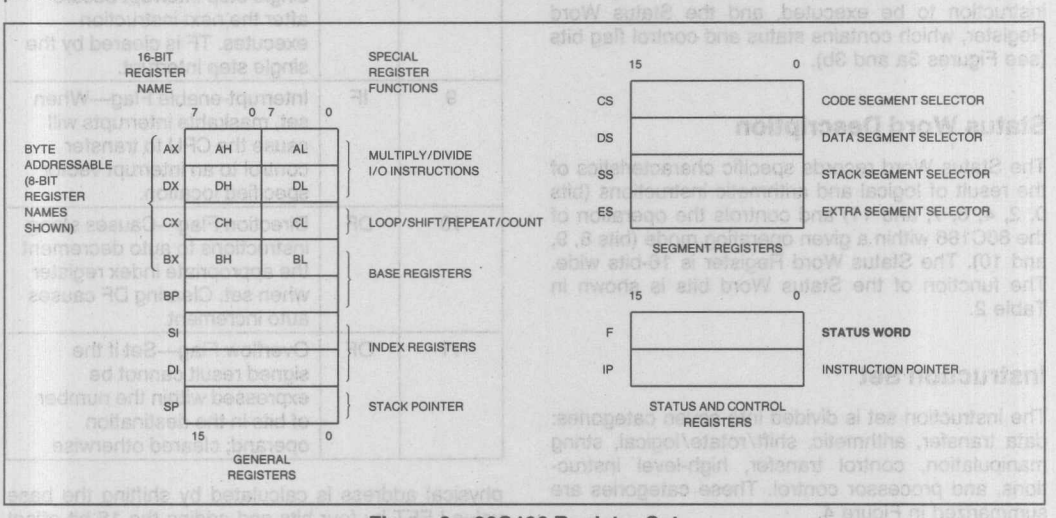


Figure 3a. 80C186 Register Set

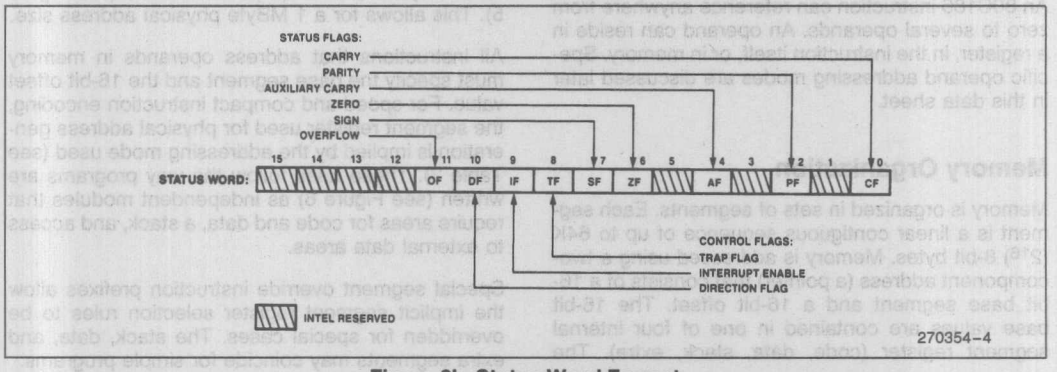


Figure 3b. Status Word Format

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80C186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80C186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80C186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80C186 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80C186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

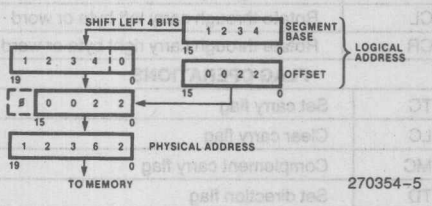


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

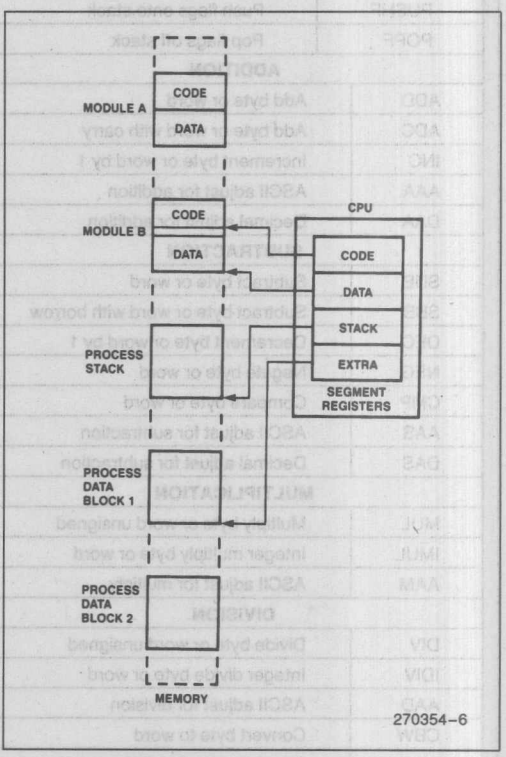


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80C186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80C186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using a Numeric Data Coprocessor with the 80C186.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using a Numeric Data Coprocessor with the 80C186.)

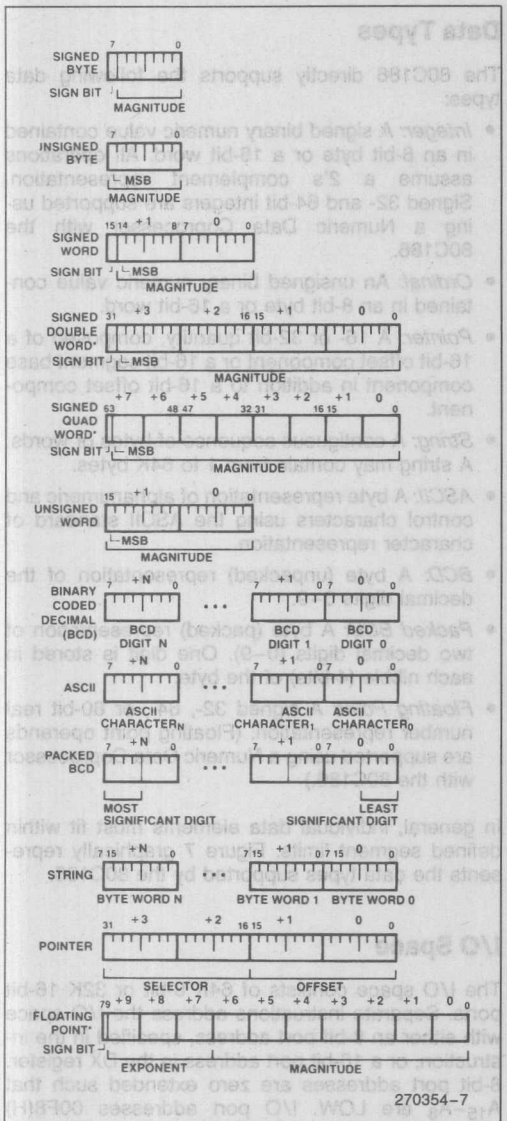
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80C186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported by using a Numeric Data Coprocessor with the 80C186.

Figure 7. 80C186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80C186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80C186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80C186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80C186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80C186 Interrupt Vectors

Interrupt Name	Vector Type	Vector Address	Default Priority	Related Instructions	Applicable Notes
Divide Error Exception	0	00H	1	DIV, IDIV	1
Single Step Interrupt	1	04H	1A	All	2
Non-Maskable Interrupt (NMI)	2	08H	1	All	
Breakpoint Interrupt	3	0CH	1	INT	1
INT0 Detected Overflow Exception	4	10H	1	INT0	1
Array Bounds Exception	5	14H	1	BOUND	1
Unused Opcode Exception	6	18H	1	Undefined Opcodes	1
ESC Opcode Exception	7	1CH	1	ESC Opcodes (Coprocessor)	1, 3
Timer 0 Interrupt	8	20H	2A		4
Timer 1 Interrupt	18	48H	2B		4
Timer 2 Interrupt	19	4CH	2C		4
Reserved	9	24H	3		
DMA 0 Interrupt	10	28H	4		
DMA 1 Interrupt	11	2CH	5		
INT0 Interrupt	12	30H	6		
INT1 Interrupt	13	34H	7		
INT2 Interrupt	14	38H	8		
INT3 Interrupt	15	3CH	9		
Numerics Coprocessor Exception	16	40H	1	ESC Opcodes (Numerics Coprocessor)	1, 5
Reserved	17	44H			
Reserved	20–31	50H . . . 7CH			

NOTES:
Default priorities for the interrupt sources are used only if the user does not program each source to a unique priority level.
1. Generated as a result of an instruction execution.
2. Performed in the same manner as 8086.
3. An ESC (coprocessor) opcode will cause a trap if the 80C186 is in compatible mode or if the processor is in enhanced mode with the proper bit set in the peripheral control block relocation register. **The 80C186 is not directly compatible with the 80186 in this respect.**
4. All three timers constitute one source of request to the interrupt controller. As such, they share the same priority level with respect to other interrupt sources. However, the timers have a defined priority order among themselves (2A > 2B > 2C).
5. Numerics coprocessor exceptions are detected by the 80C186 upon execution of a subsequent numerics instruction.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). In compatible mode operation, ESC opcodes will always generate this exception. In enhanced mode operation, the exception will be generated only if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

NOTE:
80C186 processing of ESC (numerics coprocessor) opcodes differs substantially from the 80186.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80C186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80C186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80C186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80C186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80C186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80C186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80C186 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80C186 CLOCK GENERATOR

The 80C186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The 80C186 oscillator circuit is designed to be used either with a parallel resonant fundamental or third-overtone mode crystal, depending upon the frequency range of the application as shown in Figure 8c. This is used as the time base for the 80C186. The crystal frequency chosen should be twice the required processor frequency. Use of an LC or RC circuit is not recommended.

The output of the oscillator is not directly available outside the 80C186. The two recommended crystal configurations are shown in Figure 8a. When used in third-overtone mode the tank circuit shown in Figure 8b is recommended for stable operation. The sum of the stray capacitances and loading capacitors

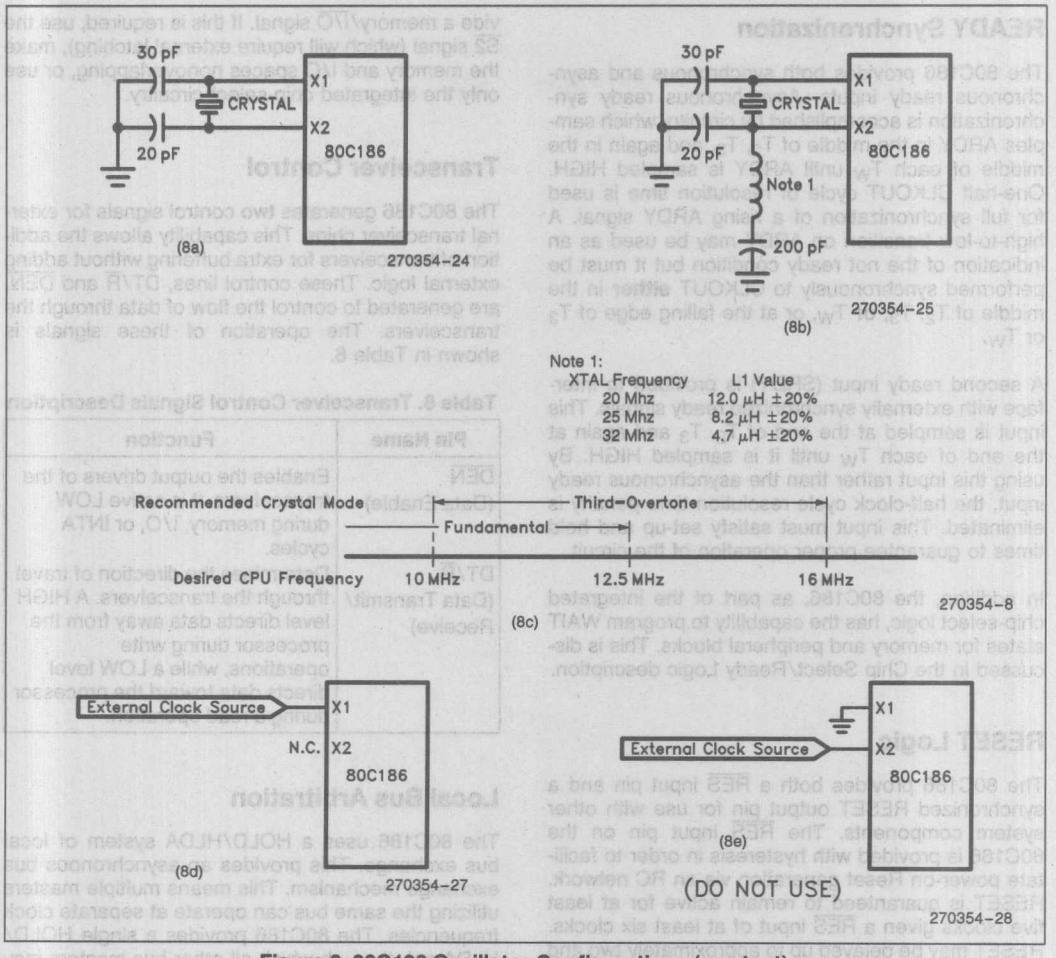


Figure 8. 80C186 Oscillator Configurations (see text)

should equal the values shown. It is advisable to limit stray capacitance between the X1 and X2 pins to less than 10 pF. While a fundamental-mode circuit will require approximately 1 ms for start-up, the third-overtone arrangement may require 1 ms to 3 ms to stabilize.

Alternately, the oscillator may be driven from an external source as shown in Figure 8d. The configuration shown in Figure 8e is not recommended.

The following parameters should be used when choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	40Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pF max
C_1 (Load Capacitance):	20 pF ± 2 pF
Drive Level:	1 mW max

Clock Generator

The 80C186 clock generator provides the 50% duty cycle processor clock for the 80C186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80C186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80C186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3, and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used for full synchronization of a rising ARDY signal. A high-to-low transition on ARDY may be used as an indication of the not ready condition but it must be performed synchronously to CLKOUT **either** in the middle of T_2, T_3, **or** T_W, or at the falling edge of T_3 or T_W.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated. This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80C186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80C186 provides both a $\overline{RES}$ input pin and a synchronized RESET output pin for use with other system components. The $\overline{RES}$ input pin on the 80C186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to approximately two and one-half clocks behind $\overline{RES}$.

LOCAL BUS CONTROLLER

The 80C186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides outputs that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80C186 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory or I/O to the 80C186 or to strobe data from the 80C186 to memory or I/O. The ALE line provides a strobe to latch the address when it is valid. The 80C186 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80C186 generates two control signals for external transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80C186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80C186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. External circuitry must arbitrate which external device will gain control of the bus when there is more than one alternate local bus master. When the 80C186 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{BHE}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80C186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is second only to DRAM refresh requests in priority of activity requests the processor may receive. Any bus cycle in progress will be completed before the 80C186 relinquishes the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency can be as great as 4 bus cycles. This will occur if a DMA word trans-

fer operation is taking place from an odd address to an odd address. This is a total of 16 clock cycles or more if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

If the 80C186 has relinquished the bus and a refresh request is pending, HLDA is removed (driven low) to signal the remote processor that the 80C186 wishes to regain control of the bus. The 80C186 will wait until HOLD is removed before taking control of the bus to run the refresh cycle.

Local Bus Controller and Reset

During RESET the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float them.

NOTE:

$\overline{RD}/\overline{QSMD}$, $\overline{UCS}$, $\overline{LCS}$, $\overline{MCS0}$/PEREQ, $\overline{MCS1}$/$\overline{ERROR}$, and $\overline{TEST}$/BUSY are provided with internal pullup devices (750Ω ±20%) which are active while $\overline{RES}$ is driven active. These devices prevent the 80C186 from entering any undesired mode of operation if the inputs are left unconnected.

- Drive $\overline{S0}$–$\overline{S2}$ to the inactive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–15, A16–19, $\overline{BHE}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80C186 integrated peripherals are controlled by 16-bit registers contained within an internal 256-byte control block. The control block may be mapped into either memory or I/O space. Internal logic will recognize control block addresses and respond to bus cycles. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80C186 CPU at any time.

The control block base address is programmed by a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select.

Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space. If the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH, which maps the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

CHIP-SELECT/READY GENERATION LOGIC

The 80C186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80C186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

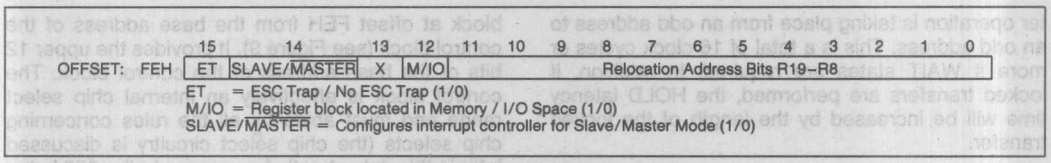


Figure 9. Relocation Register

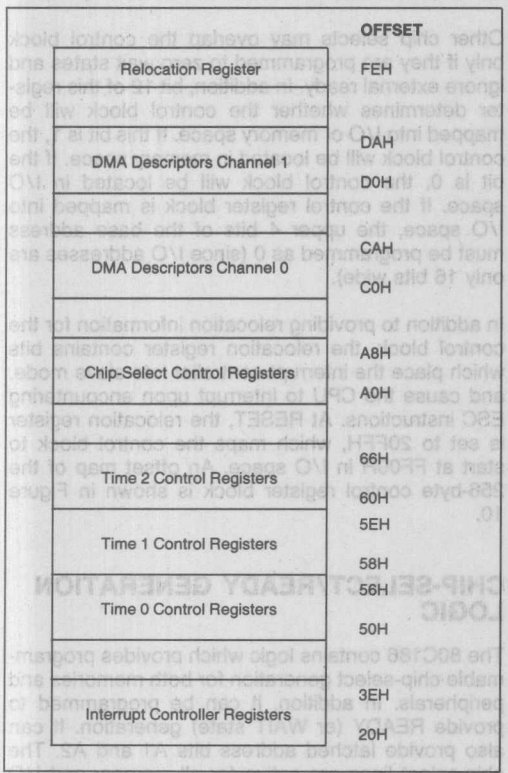


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80C186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{CS}$

The 80C186 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80C186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0 = R1 = R2 = 0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

The internal generation of any 20-bit address whose upper 16 bits are equal to or greater than the UMCS value (with bits 0–5 as "0") asserts $\overline{UCS}$. UMCS bits R2–R0 specify the ready mode for the area of memory defined by the chip select register, as explained below.

Lower Memory $\overline{CS}$

The 80C186 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0 = R1 = R2 = 0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12) at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will assert $\overline{LCS}$. LMCS register bits R2–R0 specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80C186 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located within the 80C186 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. If the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

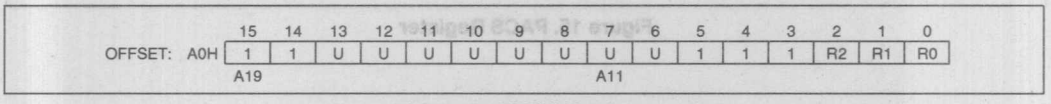


Figure 11. UMCS Register

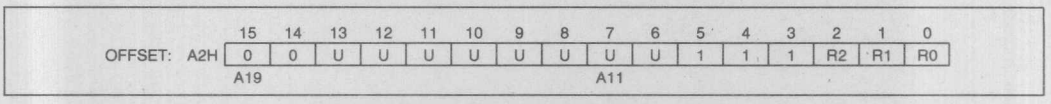


Figure 12. LMCS Register

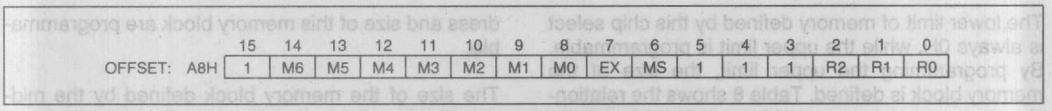


Figure 13. MPCS Register

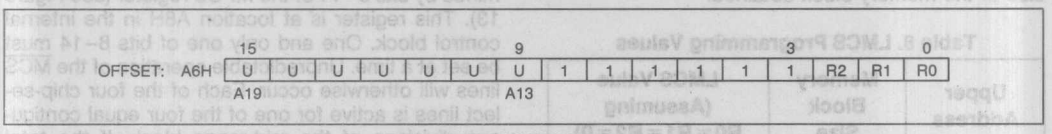


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all four mid-range chip selects.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

In Enhanced Mode, three of the four $\overline{MCS}$ pins become handshaking pins for the 80C187 Numerics Processor Extension. $\overline{MCS2}$ is still available as a chip select covering one-fourth the mid-range address block, subject to the usual programming of the MPCS and MMCS registers.

Peripheral Chip Selects

The 80C186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. The base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80C186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1 and A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0 and A1 pins used for selecting internal registers of external 8-bit peripheral chips. This scheme simplifies the external hardware because the peripheral registers can be located on even boundaries in I/O or memory space.

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). The register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

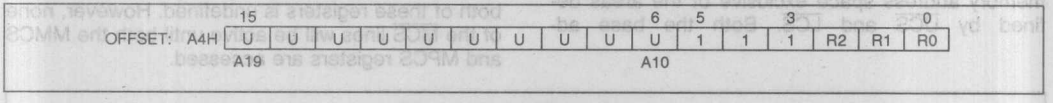


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

$\overline{PCS}$ Line	Active between Locations
$\overline{PCS0}$	PBA —PBA + 127
$\overline{PCS1}$	PBA + 128—PBA + 255
$\overline{PCS2}$	PBA + 256—PBA + 383
$\overline{PCS3}$	PBA + 384—PBA + 511
$\overline{PCS4}$	PBA + 512—PBA + 639
$\overline{PCS5}$	PBA + 640—PBA + 767
$\overline{PCS6}$	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). The register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 specify the READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80C186 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80C186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80C186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). For example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon RESET, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to insert 3 wait states in conjunction with external READY (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

The 80C186 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data.

DMA Operation

Each channel has six registers in the control block which define each channel's operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Count Register, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

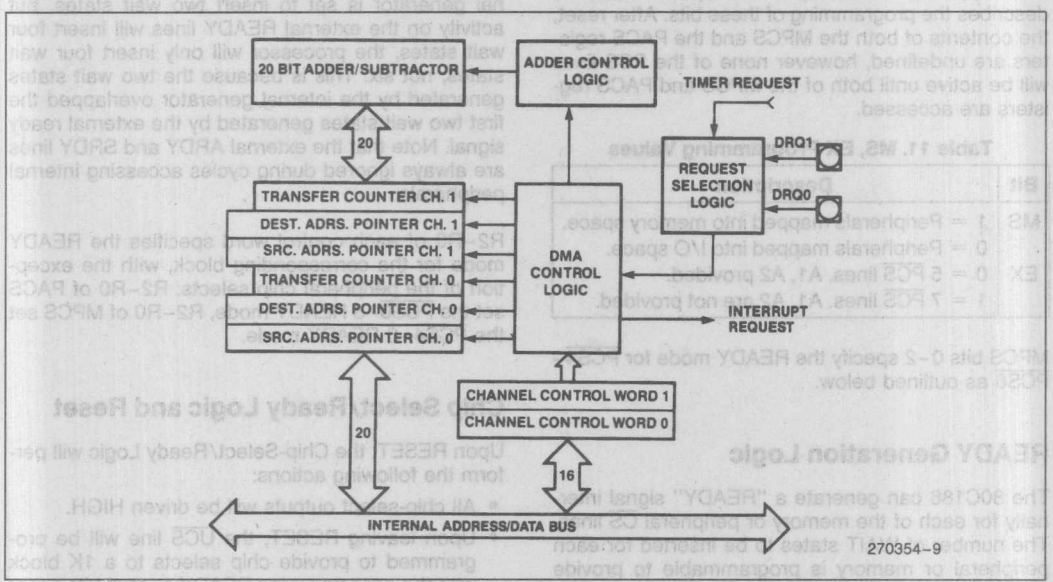


Figure 16. DMA Unit Block Diagram

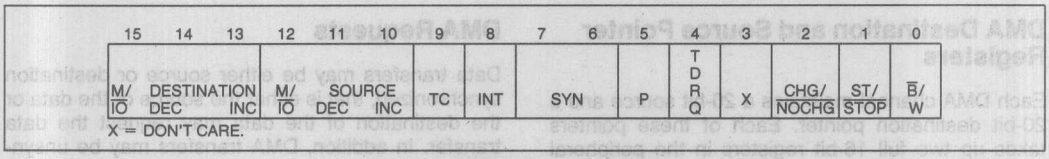


Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80C186 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

DEST: M/$\overline{IO}$ Destination pointer is in memory (1) and I/O (0) space.

DEC Decrement destination pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

INC Increment destination pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

If both INC and DEC are specified, the pointer will not be changed after each cycle.

SOURCE: M/$\overline{IO}$ Source pointer is in memory (1) or I/O (0) space.

DEC Decrement source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

INC Increment source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

If both INC and DEC are specified, the pointer will remain constant after each cycle.

TC: If set, DMA will terminate when the contents of the transfer count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point. If cleared, the DMA controller will decrement the transfer count register for each DMA cycle, but DMA transfers will not stop when the transfer count register reaches zero.

INT: Enable interrupts to CPU upon transfer count termination.

SYN: 00 No synchronization.

NOTE:
When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

P: Channel priority relative to other channel.

0 Low priority.

1 High priority.

Channels will alternate cycles if both are set at same priority level.

TDRQ: Enable/Disable (1/0) DMA requests from timer 2.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) ST/$\overline{STOP}$ bit. If this bit is set when writing to the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be read as 0.

ST/$\overline{STOP}$: Start/Stop (1/0) channel.

$\overline{B}$/W: Byte/Word (0/1) transfers.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. For each DMA channel to be used, all four pointer registers must be initialized. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two.

Each pointer may point into either memory or I/O space. Since the upper four bits of the address are not automatically programmed to zero, the user must program them in order to address the normal 64K I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be achieved if all word transfers are performed to or from even addresses so that accesses will occur in single bus cycles.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). The register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). When destination synchronization is performed, data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This allows the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. Table 14 shows the maximum DMA transfer rates.

Table 14. Maximum DMA Transfer Rates at 16 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	4.0MBytes/sec	4.0MBytes/sec
Source Synch	4.0MBytes/sec	4.0MBytes/sec
Destination Synch	2.7MBytes/sec	3.2MBytes/sec

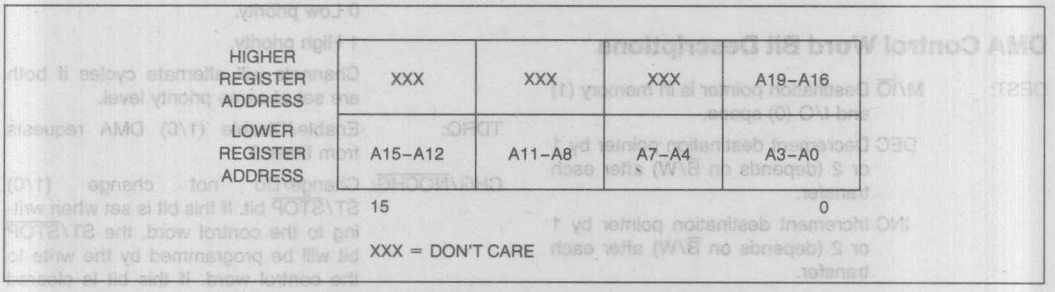


Figure 18. DMA Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed to give one channel priority over the other, or they may be programmed to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; also an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/$\overline{\text{STOP}}$ bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also be generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before the ST/$\overline{\text{STOP}}$ bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/$\overline{\text{STOP}}$ bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the state of the DMA channels will be as follows:

- The ST/$\overline{\text{STOP}}$ bit for each channel will be reset to $\overline{\text{STOP}}$.
- Any transfer in progress is aborted.
- The values of the transfer count registers, source pointers, and destination pointers are indeterminate.

TIMERS

The 80C186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, the third timer can be used as a prescaler to the other two, or as a DMA request source.

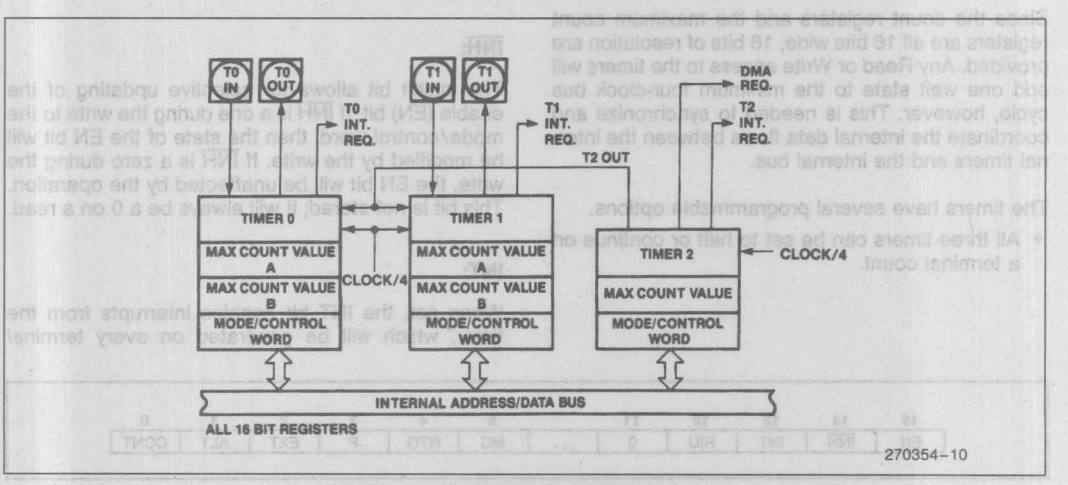


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate. Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{INH}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{INH}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{INH}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal

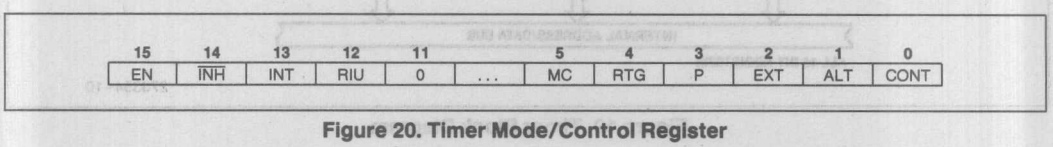


Figure 20. Timer Mode/Control Register

count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80C186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80C186 clock.

If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The contents of this register may be read or written by the processor at any time. If the register is written while the timer is counting, the new value will take effect in the current count cycle.

The count registers should be programmed before attempting to use the timers since they are not automatically initialized to zero.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. A timer resets when the timer count register equals the max count value being used. If the timer count register or the max count register is changed so that the max count is less than the timer count, the timer does not immediately reset. Instead, the timer counts up to 0FFFFH, "wraps around" to zero, counts up to the max count value, and then resets.

Timers and Reset

Upon RESET, the state of the timers will be as follows:

- All EN (Enable) bits are reset preventing timer counting.
- For Timers 0 and 1, the RIU bits are reset to zero and the ALT bits are set to one. This results in the Timer Out pins going HIGH.
- The contents of the count registers are indeterminate.

INTERRUPT CONTROLLER

The 80C186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80C186 interrupt controller has its own control register that sets the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80C186 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

Five pins are provided for external interrupt sources. One of these pins is NMI, the non-maskable interrupt. NMI is generally used for unusual events such as power-fail interrupts. The other four pins may be configured in any of the following ways:

- As four interrupt lines with internally generated interrupt vectors.
- As an interrupt line and interrupt acknowledge line pair (Cascade Mode) with externally generated interrupt vectors plus two interrupt input lines with internally generated vectors.
- As two pairs of interrupt/interrupt acknowledge lines (Cascade Mode) with externally generated interrupt vectors.

External sources in the Cascade Mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{INTA}$ cycles are initiated and the vector is read into the 80C186 on the second cycle. The capability to interface to external 82C59A programmable interrupt controllers is provided when the inputs are configured in Cascade Mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 82C59A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled, yet be suspended only by interrupts of higher priority than the in-service interrupt.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is executed at the end of the service routine just before the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80C186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the Cascade Mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 82C59A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the Cascade or non-Cascade Mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary Cascade Mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 82C59As. Three levels of priority are created, requiring priority resolution in the 80C186 interrupt controller, the master 82C59As, and the slave 82C59As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

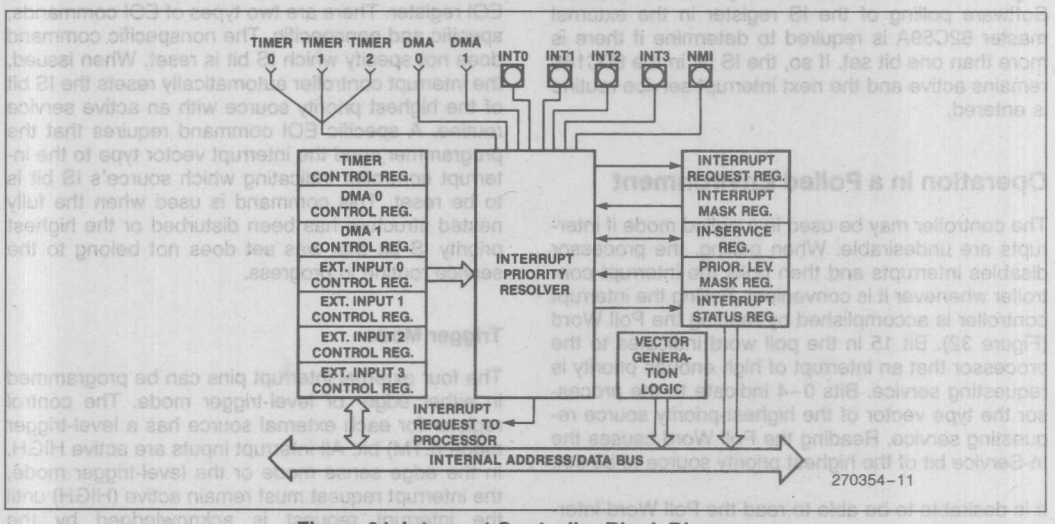


Figure 21. Interrupt Controller Block Diagram

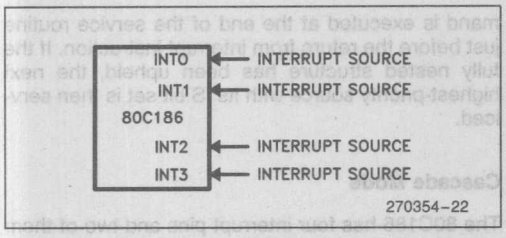


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 82C59A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80C186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80C186 controller until the 80C186 in-service bit is reset. In special fully nested mode, the 80C186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80C186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the IS register in the external master 82C59A is required to determine if there is more than one bit set. If so, the IS bit in the 80C186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80C186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, other interrupt requests can be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the

80C186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80C186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits are set when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corre-

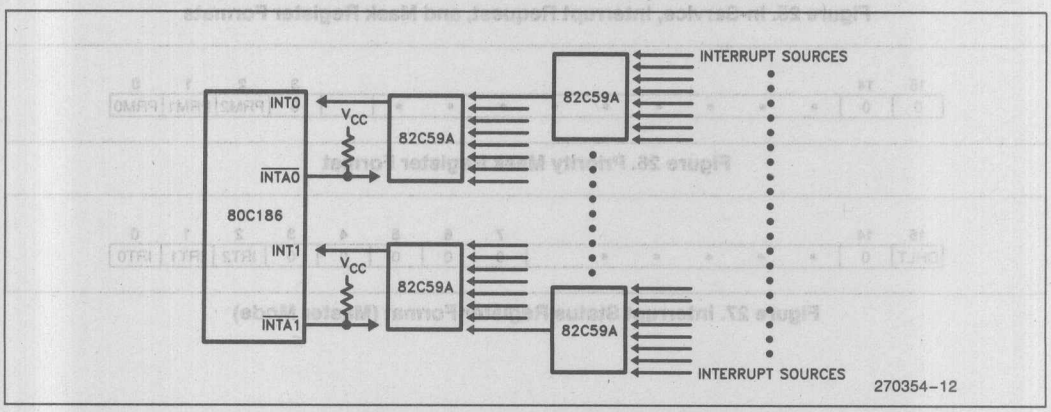


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

sponding to a particular source masks the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

15	14	•	•	•	10	9	8	7	6	5	4	3	2	1	0
0	0				0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14	•	•	•	•	•	•	•	•	•	3	2	1	0
0	0										0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14	•	•	•	•	•	7	6	5	4	3	2	1	0
DHLT	0						0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Priority Mask Register

This register masks all interrupts below a particular interrupt priority level. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. This bit allows prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits differentiate between timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80C186 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

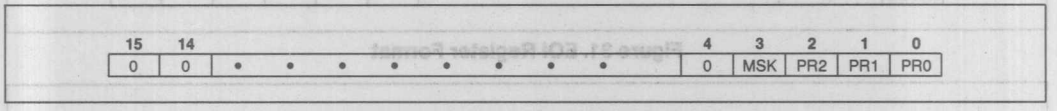


Figure 28. Timer/DMA Control Registers Formats

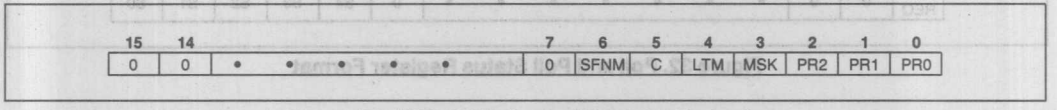


Figure 29. INT0/INT1 Control Register Formats

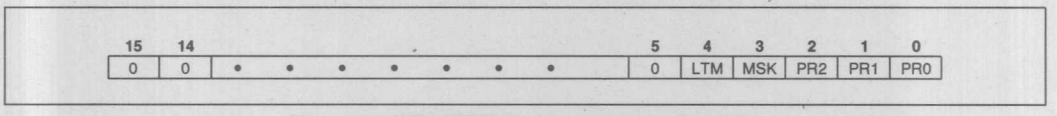


Figure 30. INT2/INT3 Control Register Formats

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80C186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80C186 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80C186 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 82C59A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80C186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

Slave Mode External Interface

The configuration of the 80C186 with respect to an external 82C59A master is shown in Figure 33. The INT0 (Pin 45) input is used as the 80C186 CPU interrupt input. INT3 (Pin 41) functions as an output to send the 80C186 slave-interrupt-request to one of the 8 master-PIC-inputs.

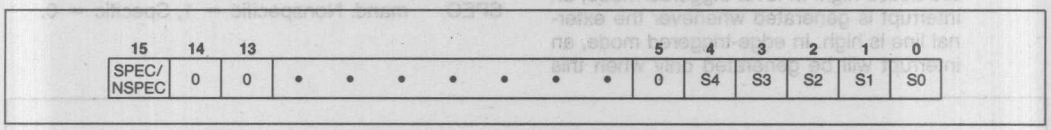


Figure 31. EOI Register Format

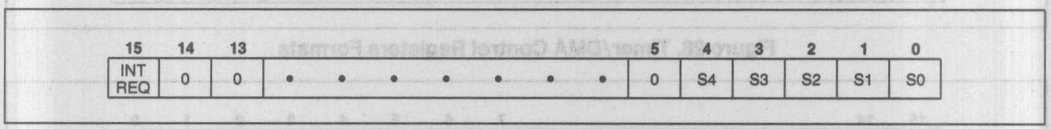


Figure 32. Poll and Poll Status Register Format

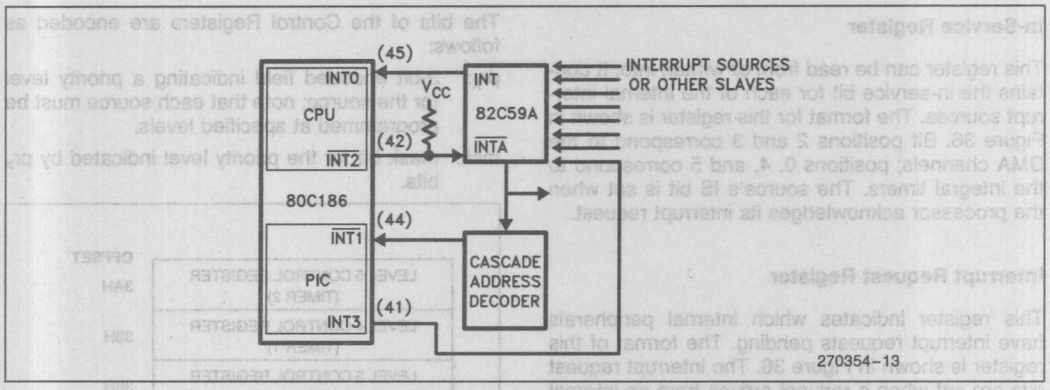


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 82C59As do this internally. Because of pin limitations, the 80C186 slave address will have to be decoded externally. $\overline{INT1}$ (Pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ (Pin 42) is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 82C59A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 82C59A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80C186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented (see Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

Enhanced Mode Operation

In Compatible Mode the 80C186 operates with all the features of the NMOS 80186, with the exception of 8087 support (i.e. no numeric coprocessing is possible in Compatible Mode). Queue-Status information is still available for design purposes other than 8087 support.

All the Enhanced Mode features are completely masked when in Compatible Mode. A write to any of the Enhanced Mode registers will have no effect, while a read will not return any valid data.

In Enhanced Mode, the 80C186 will operate with Power-Save, DRAM refresh, and numerics coprocessor support in addition to all the Compatible Mode features.

Entering Enhanced Mode

If connected to a numerics coprocessor, this mode will be invoked automatically. Without a NPX, this mode can be entered by tying the RESET output signal from the 80C186 to the TEST/BUSY input.

Queue-Status Mode

The queue-status mode is entered by strapping the $\overline{RD}$ pin low. $\overline{RD}$ is sampled at RESET and if LOW, the 80C186 will reconfigure the ALE and $\overline{WR}$ pins to be QS0 and QS1 respectively. This mode is available on the 80C186 in both Compatible and Enhanced Modes.

DRAM Refresh Control Unit Description

The Refresh Control Unit (RCU) automatically generates DRAM refresh bus cycles. The RCU operates only in Enhanced Mode. After a programmable period of time, the RCU generates a memory read request to the BIU. If the address generated during a refresh bus cycle is within the range of a properly programmed chip select, that chip select will be activated when the BIU executes the refresh bus cycle. The ready logic and wait states programmed for that region will also be in force. If no chip select is activated, then external ready is automatically required to terminate the refresh bus cycle.

If the HLDA pin is active when a DRAM refresh request is generated (indicating a bus hold condition), then the 80C186 will deactivate the HLDA pin in order to perform a refresh cycle. The circuit external to the 80C186 must remove the HOLD signal in order to execute the refresh cycle. The sequence of HLDA going inactive while HOLD is being held active can be used to signal a pending refresh request.

All registers controlling DRAM refresh may be read and written in Enhanced Mode. When the processor is operating in Compatible Mode, they are deselected and are therefore inaccessible. Some fields of these registers cannot be written and are always read as zeros.

DRAM Refresh Addresses

The address generated during a refresh cycle is determined by the contents of the MDRAM register (see Figure 40) and the contents of a 9-bit counter. Figure 41 illustrates the origin of each bit.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MDRAM: Offset E0H	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0

Bits 0–8: Reserved, read back as 0.

Bits 9–15: M0–M6, are address bits A13–A19 of the 20-bit memory refresh address. These bits should correspond to the chip select address to be activated for the DRAM partition. These bits are set to 0 on RESET.

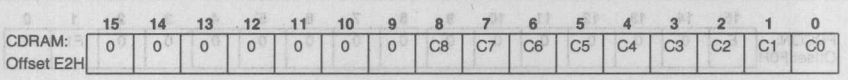
Figure 40. Memory Partition Register

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
M6	M5	M4	M3	M2	M1	M0	0	0	0	CA8	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0	1

M6–M0: Bits defined by MDRAM Register

CA8–CA0: Bits defined by refresh address counter. Note, these bits do not directly follow a binary count. Rather they are implemented using a Linear/Feedback shift register.

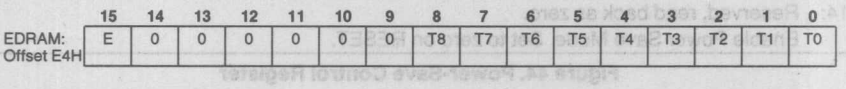
Figure 41. Addresses Generated by RCU



Bits 0–8: C0–C8, clock divisor register, holds the number of CLKOUT cycles between each refresh request.

Bits 9–15: Reserved, read back as 0.

Figure 42. Clock Pre-Scaler Register



Bits 0–8: T0–T8, refresh clock counter outputs. Read only.

Bits 9–14: Reserved, read back as 0.

Bit 15: Enable RCU, set to 0 on RESET.

Figure 43. Enable RCU Register

Refresh Control Unit Programming and Operation

After programming the MDRAM and the CDRAM registers (Figures 40 and 42), the RCU is enabled by setting the "E" bit in the EDRAM register (Figure 43). The clock counter (T0–T8 of EDRAM) will be loaded from C0–C8 of CDRAM during T_3 of instruction cycle that sets the "E" bit. The clock counter is then decremented at each subsequent CLKOUT.

A refresh is requested when the value of the counter has reached 1 and the counter is reloaded from CDRAM. In order to avoid missing refresh requests, the value in the CDRAM register should always be at least 18 (12H). Clearing the "E" bit at anytime will clear the counter and stop refresh requests, but will not reset the refresh address counter.

POWER-SAVE CONTROL

Power Save Operation

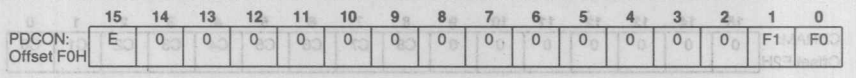
The 80C186, when in Enhanced Mode, can enter a power saving state by internally dividing the clock-in frequency by a programmable factor. This divided frequency is also available at the CLKOUT pin. The PDCON register contains the two-bit fields for selecting the clock division factor and the enable bit.

All internal logic, including the Refresh Control Unit and the timers, will have their clocks slowed down by the division factor. To maintain a real time count or a fixed DRAM refresh rate, these peripherals must be re-programmed when entering and leaving the power-save mode.

The power-save mode is exited whenever an interrupt is processed by automatically resetting the enable bit. If the power-save mode is to be re-entered after serving the interrupt, the enable bit will need to be set in software before returning from the interrupt routine.

The internal clocks of the 80C186 will begin to be divided during the T_3 state of the instruction cycle that sets the enable bit. Clearing the enable bit will restore full speed in the T_3 state of that instruction.

At no time should the internal clock frequency be allowed to fall below 0.5 MHz. This is the minimum operational frequency of the 80C186. For example, an 80C186 running with a 12 MHz crystal (6 MHz CLOCKOUT) should never have a clock divisor greater than eight.



Bits 0–1: Clock Divisor Select

F1	F0	Division Factor
0	0	divide by 1
0	1	divide by 4
1	0	divide by 8
1	1	divide by 16

Bits 2–14: Reserved, read back as zero.

Bit 15: Enable Power Save Mode. Set to zero on RESET.

Figure 44. Power-Save Control Register

Interface for 80C187 Numeric Processor Extension

In Enhanced Mode, three of the mid-range memory chip selects are redefined according to Table 16 for use with the 80C187. The fourth chip select, $\overline{MCS2}$ functions as in compatible mode, and may be programmed for activity with ready logic and wait states accordingly. As in compatible mode, $\overline{MCS2}$ will function for one-fourth a programmed block size.

Table 16. $\overline{MCS}$ Assignments

Compatible Mode	Enhanced Mode	
$\overline{MCS0}$	PEREQ	Processor Extension Request
$\overline{MCS1}$	$\overline{ERROR}$	NPX Error
$\overline{MCS2}$	$\overline{MCS2}$	Mid-Range Chip Select
$\overline{MCS3}$	$\overline{NPS}$	Numeric Processor Select

Four port addresses are assigned to the 80C186/80C187 interface for 16-bit reads and writes. Table 17 shows the port definitions. These ports are not accessible by using the 80C186 I/O instructions. However, numerics operations will cause a $\overline{PCS}$ line to be activated if it is properly programmed for this I/O range.

Table 17. Numerics Coprocessor I/O Port Assignments

I/O Address	Read Definition	Write Definition
00F8H	Status/Control	Opcode
00FAH	Data	Data
00FCH	reserved	CS:IP, DS:EA
00FEH	Opcode Status	reserved

ONCE™ Test Mode

To facilitate testing and inspection of devices when fixed into a target system, the 80C186 has a test mode available which allows all pins to be placed in a high-impedance state. ONCE stands for "ON Circuit Emulation". When placed in this mode, the 80C186 will put all pins in the high-impedance state until RESET.

The ONCE mode is selected by tying the $\overline{UCS}$ and the $\overline{LCS}$ LOW during RESET. These pins are sampled on the low-to-high transition of the $\overline{RES}$ pin. The $\overline{UCS}$ and the $\overline{LCS}$ pins have weak internal pull-up resistors similar to the $\overline{RD}$ and $\overline{TEST}$/BUSY pins to guarantee normal operation.

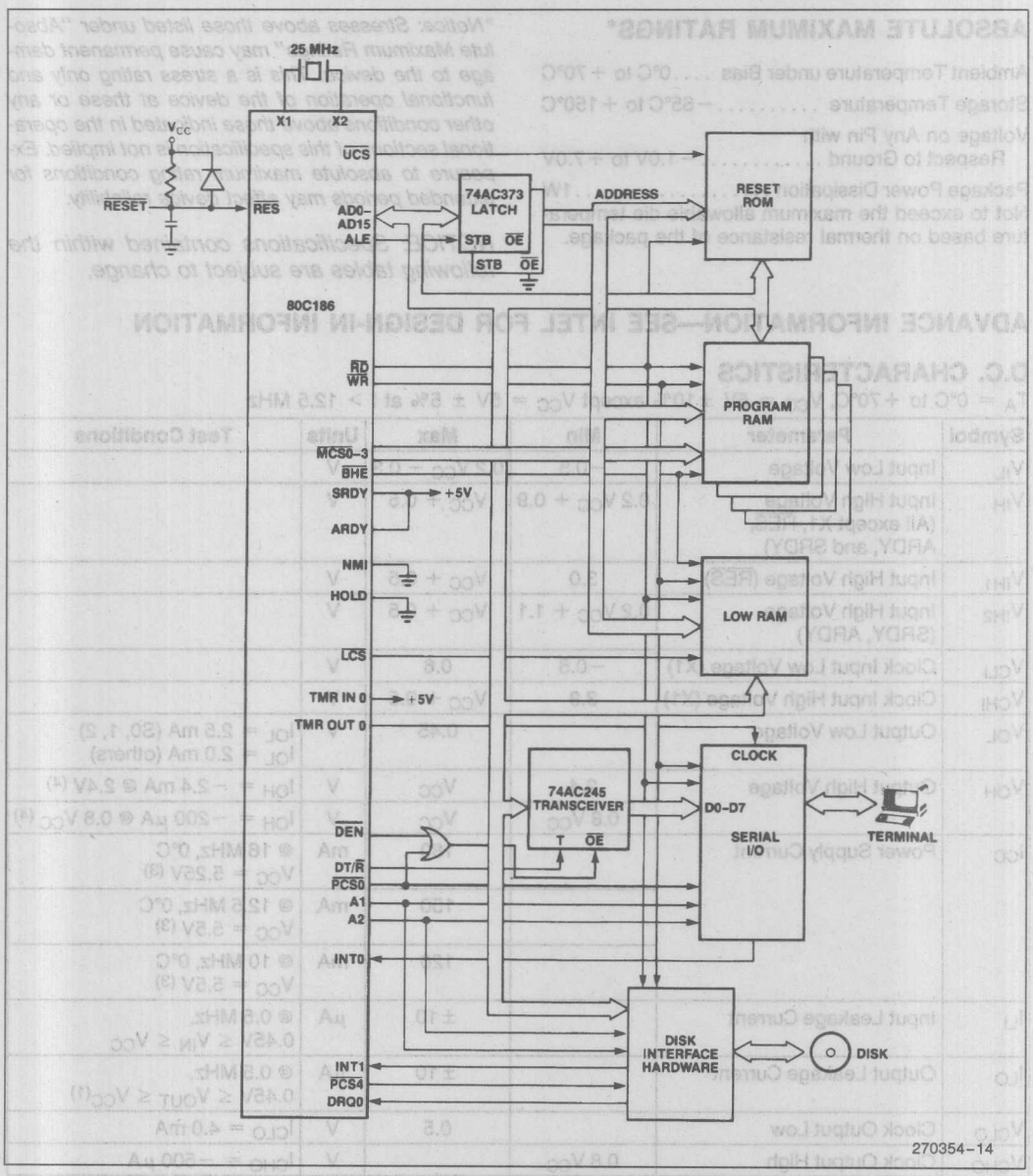


Figure 45. Typical 80C186 Computer

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C

Storage Temperature .....−65°C to +150°C

Voltage on Any Pin with Respect to Ground .....−1.0V to +7.0V

Package Power Dissipation 1W
Not to exceed the maximum allowable die temperature based on thermal resistance of the package.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ± 5% at f > 12.5 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.2 V_{CC} − 0.3	V	
V_{IH}	Input High Voltage (All except X1, $\overline{RES}$, ARDY, and SRDY)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{IH2}	Input High Voltage (SRDY, ARDY)	0.2 V_{CC} + 1.1	V_{CC} + 0.5	V	
V_{CLI}	Clock Input Low Voltage (X1)	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage (X1)	3.9	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA (S0, 1, 2) I_{OL} = 2.0 mA (others)
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −2.4 mA @ 2.4V (4)
		0.8 V_{CC}	V_{CC}	V	I_{OH} = −200 µA @ 0.8 V_{CC} (4)
I_{CC}	Power Supply Current		180	mA	@ 16 MHz, 0°C V_{CC} = 5.25V (3)
			150	mA	@ 12.5 MHz, 0°C V_{CC} = 5.5V (3)
			120	mA	@ 10 MHz, 0°C V_{CC} = 5.5V (3)
I_{LI}	Input Leakage Current		±10	µA	@ 0.5 MHz, 0.45V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	µA	@ 0.5 MHz, 0.45V ≤ V_{OUT} ≤ V_{CC}(1)
V_{CLO}	Clock Output Low		0.5	V	I_{CLO} = 4.0 mA
V_{CHO}	Clock Output High	0.8 V_{CC}		V	I_{CHO} = −500 µA
C_{IN}	Input Capacitance		10	pF	@ 1 MHz(2)
C_{IO}	I/O Capacitance		20	pF	@ 1 MHz(2)

NOTES:
1. Pins being floated during HOLD or by invoking the ONCE Mode.
2. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at + 5.0V or 0.45V. This parameter is not tested.
3. Current is measured with the device in RESET with X1 and X2 driven and all other non-power pins open.
4. $\overline{RD}$/QSMD, $\overline{UCS}$, $\overline{LCS}$, $\overline{MSC0}$/PEREQ, $\overline{MCS1}$/ERROR, and $\overline{TEST}$/BUSY pins have internal pullup devices that are active at RESET. Excessive loading on these pins can cause the 80C186 to go into undesired modes of operation (e.g., Queue Status, ONCE) upon RESET.

POWER SUPPLY CURRENT

Current is linearly proportional to clock frequency and is measured with the device in RESET with X1 and X2 driven and all other non-power pins open.

Maximum current is indicated in the D.C. Characteristics.

Typical current is given by I_{CC} (typical) = 6.4 mA × freq. (MHz) + 4.0 mA. "Typicals" are based on a limited number of samples taken from early manufacturing lots measured at V_{CC} = 5V and room temperature. "Typicals" are not guaranteed.

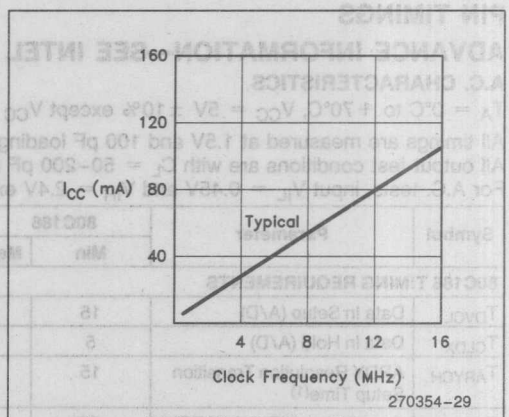


Figure 46. I_{CC} (typical) vs Frequency

PIN TIMINGS

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} − 0.5V

Symbol	Parameter	80C186		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 TIMING REQUIREMENTS									
T_{DVCL}	Data In Setup (A/D)	15		15		10		ns	
T_{CLDX}	Data In Hold (A/D)	5		5		5		ns	
T_{ARYCH}	ARDY Resolution Transition Setup Time(1)	15		15		15		ns	
T_{ARYLCL}	Asynchronous Ready (ARDY) Setup Time	25		25		25		ns	
T_{CLARX}	ARDY Active Hold Time	15		15		15		ns	
T_{ARYCHL}	ARDY Inactive Hold Time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time (2)	15		15		15		ns	
T_{CLSRY}	SRDY Transition Hold Time (2)	15		15		15		ns	
T_{HVCL}	HOLD Setup(1)	15		15		15		ns	
T_{INVCH}	INTR, NMI, TEST, TMR IN Setup Time(1)	15		15		15		ns	
T_{INVCL}	DRQ0, DRQ1, $\overline{RES}$, Setup Time(1)	15		15		15		ns	
80C186 MASTER INTERFACE TIMING RESPONSES									
T_{CLAV}	Address Valid Delay	5	50	5	36	5	33	ns	C_L=50 pF −200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLAX}	Address Hold	0		0		0		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	30	T_{CLAX}	25	T_{CLAX}	20	ns	
T_{CHCZ}	Command Lines Float Delay		40		33		28	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		45		37		32	ns	
T_{LHLL}	ALE Width (min)	T_{CLCL} − 30		T_{CLCL} − 30		T_{CLCL} − 30		ns	C_L=50 pF −100 pF all outputs @ 12.5 & 16 MHz
T_{CHLH}	ALE Active Delay		30		25		20	ns	
T_{CHLL}	ALE Inactive Delay		30		25		20	ns	
T_{LLAX}	Address Hold to ALE Inactive (min)	T_{CHCL} − 20		T_{CHCL} − 15		T_{CHCL} − 15		ns	
T_{CLDV}	Data Valid Delay	5	40	5	36	5	33	ns	
T_{CLDOX}	Data Hold Time	3		3		3		ns	
T_{WHDX}	Data Hold after $\overline{WR}$ (min)	T_{CLCL} − 34		T_{CLCL} − 20		T_{CLCL} − 20		ns	
T_{CVCTV}	Control Active Delay 1	3	56	3	47	3	31	ns	
T_{CHCTV}	Control Active Delay 2	5	44	5	37	5	31	ns	
T_{CVCTX}	Control Inactive Delay	3	44	3	37	3	31	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	5	56	5	47	5	35	ns	

NOTE:
1. To guarantee recognition at next clock.
2. To guarantee proper operation.

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} − 0.5V

Symbol	Parameter	80C186		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 MASTER INTERFACE TIMING RESPONSES (Continued)									
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		0		ns	C_L=50–200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLRL}	$\overline{RD}$ Active Delay	5	44	5	37	5	31	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	5	44	5	37	5	31	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active (min)	T_{CLCL} − 40		T_{CLCL} − 20		T_{CLCL} − 20		ns	C_L=50–100 pF all outputs @ 12.5 & 16 MHz
T_{CLHAV}	HLDA Valid Delay	3	40	3	33	3	25	ns	
T_{RLRH}	$\overline{RD}$ Pulse Width (min)	$2T_{CLCL}$ − 46		$2T_{CLCL}$ − 40		$2T_{CLCL}$ − 30		ns	
T_{WLWH}	$\overline{WR}$ Pulse Width (min)	$2T_{CLCL}$ − 34		$2T_{CLCL}$ − 30		$2T_{CLCL}$ − 25		ns	
T_{AVLL}	Address Valid to ALE Low (min)	T_{CLCH} − 19		T_{CLCH} − 15		T_{CLCH} − 15		ns	Equal Loading
T_{CHSV}	Status Active Delay	5	45	5	35	5	31	ns	
T_{CLSH}	Status Inactive Delay	5	50	5	35	5	30	ns	
T_{CLTMV}	Timer Output Delay		48		40		30	ns	100 pF max @ 10 MHz
T_{CLRO}	Reset Delay		48		40		30	ns	C_L = 50–200 pF All outputs (except T_{CLTMV}) @ 10 MHz
T_{CHQSV}	Queue Status Delay		28		28		25	ns	
T_{CHDX}	Status Hold Time	5		5		5		ns	
T_{AVCH}	Address Valid to Clock High	0		0		0		ns	C_L = 50–100 pF All outputs @ 12.5 & 16 MHz
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	3	45	3	40	3	35	ns	
T_{DXDL}	$\overline{DEN}$ Inactive to DT/$\overline{R}$ Low	0		0		0		ns	Equal Loading
80C186 CHIP-SELECT TIMING RESPONSES									
T_{CLCSV}	Chip-Select Active Delay		45		33		30	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	T_{CLCH} − 10		T_{CLCH} − 10		T_{CLCH} − 10		ns	Equal Loading
T_{CHCSX}	Chip-Select Inactive Delay	5	32	5	28	5	23	ns	

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} − 0.5V

Symbol	Parameter	80C186		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 CLKIN REQUIREMENTS Measurements taken with following conditions: External clock input to X1 and X2 not connected (float)									
T_{CKIN}	CLKIN Period	50	1000	40	1000	31.25	1000	ns	
T_{CKHL}	CLKIN Fall Time		5		5		5	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		5		5		5	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	20		16		13		ns	1.5V(2)
T_{CHCK}	CLKIN High Time	20		16		13		ns	1.5V(2)
80C186 CLKOUT TIMING 200 pF load maximum for 10 MHz or less, 100 pF load maximum above 10 MHz									
T_{CICO}	CLKIN to CLKOUT Skew		25		21		17	ns	
T_{CLCL}	CLKOUT Period	100	2000	80	2000	62.5	2000	ns	
T_{CLCH}	CLKOUT Low Time (min)	0.5 T_{CLCL}−8		0.5 T_{CLCL}−7		0.5 T_{CLCL}−7		ns	C_L = 100 pF (2)
		0.5 T_{CLCL}−6		0.5 T_{CLCL}−5		0.5 T_{CLCL}−5		ns	C_L = 50 pF (3)
T_{CHCL}	CLKOUT High Time (min)	0.5 T_{CLCL}−8		0.5 T_{CLCL}−7		0.5 T_{CLCL}−7		ns	C_L = 100 pF (4)
		0.5 T_{CLCL}−6		0.5 T_{CLCL}−5		0.5 T_{CLCL}−5		ns	C_L = 50 pF (3)
T_{CH1CH2}	CLKOUT Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		10		10		8	ns	3.5 to 1.0V

NOTES:
1. T_{CLCK} and T_{CHCK} (CLKIN Low and High times) should not have a duration less than 40% of T_{CKIN}.
2. Tested under worst case conditions: V_{CC} = 5.5V (5.25V @ 16 MHz), T_A = 70°C.
3. Not tested.
4. Tested under worst case conditions: V_{CC} = 4.5V (4.75V @ 16 MHz), T_A = 0°C.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has from 5 to 7 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

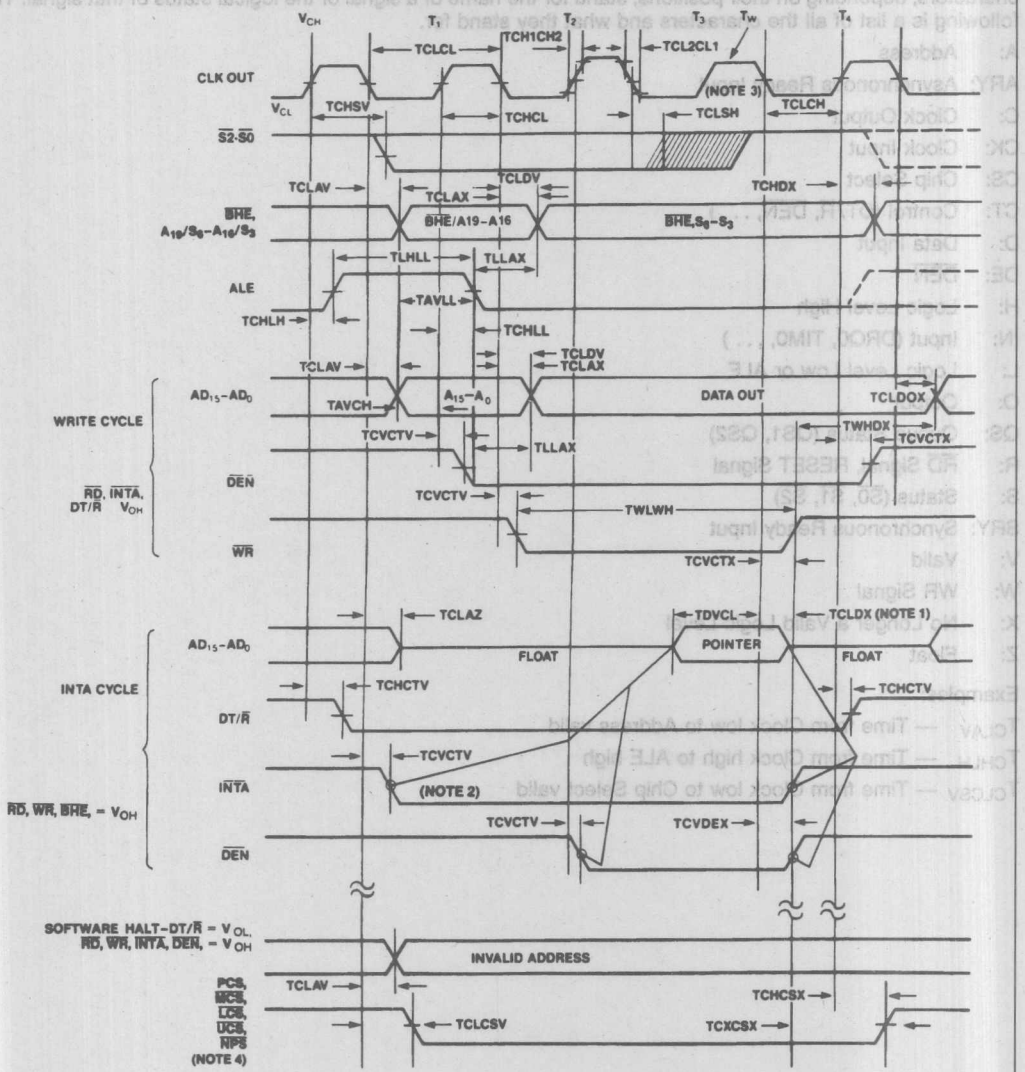
A: Address
ARY: Asynchronous Ready Input
C: Clock Output
CK: Clock Input
CS: Chip Select
CT: Control (DT/$\overline{R}$, $\overline{DEN}$, . . .)
D: Data Input
DE: $\overline{DEN}$
H: Logic Level High
IN: Input (DRQ0, TIM0, . . .)
L: Logic Level Low or ALE
O: Output
QS: Queue Status (QS1, QS2)
R: $\overline{RD}$ Signal, RESET Signal
S: Status ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$)
SRY: Synchronous Ready Input
V: Valid
W: WR Signal
X: No Longer a Valid Logic Level
Z: Float

Examples:

T_{CLAV} — Time from Clock low to Address valid
T_{CHLH} — Time from Clock high to ALE high
T_{CLCSV} — Time from Clock low to Chip Select valid

WAVEFORMS

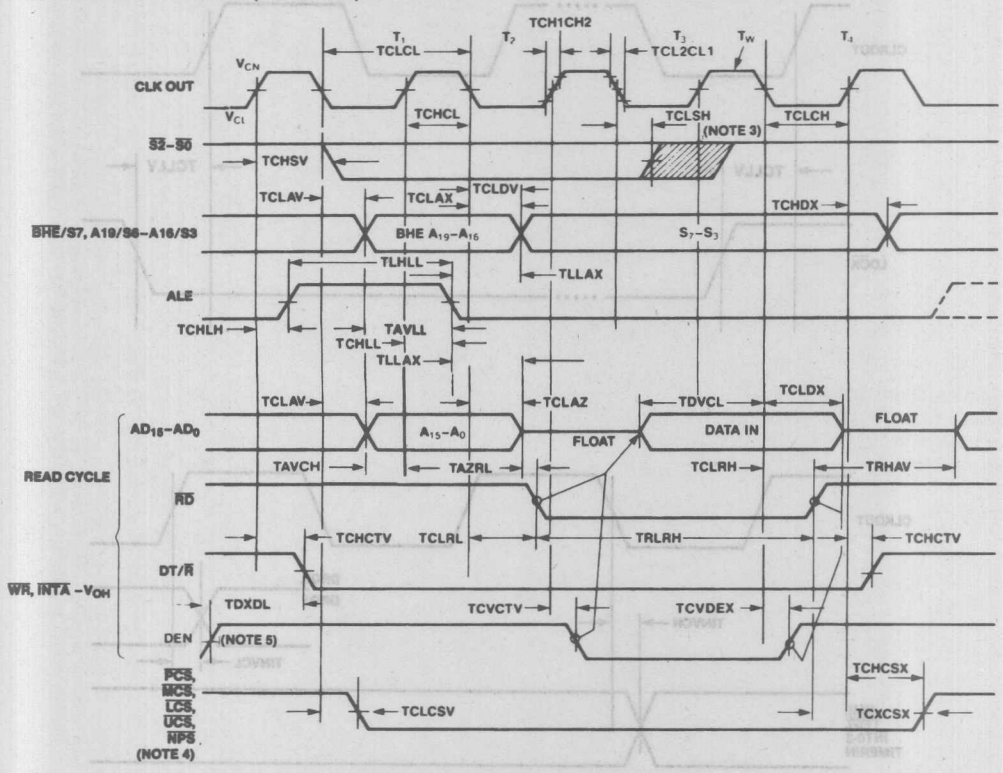
MAJOR CYCLE TIMING



270354-16

WAVEFORMS (Continued)

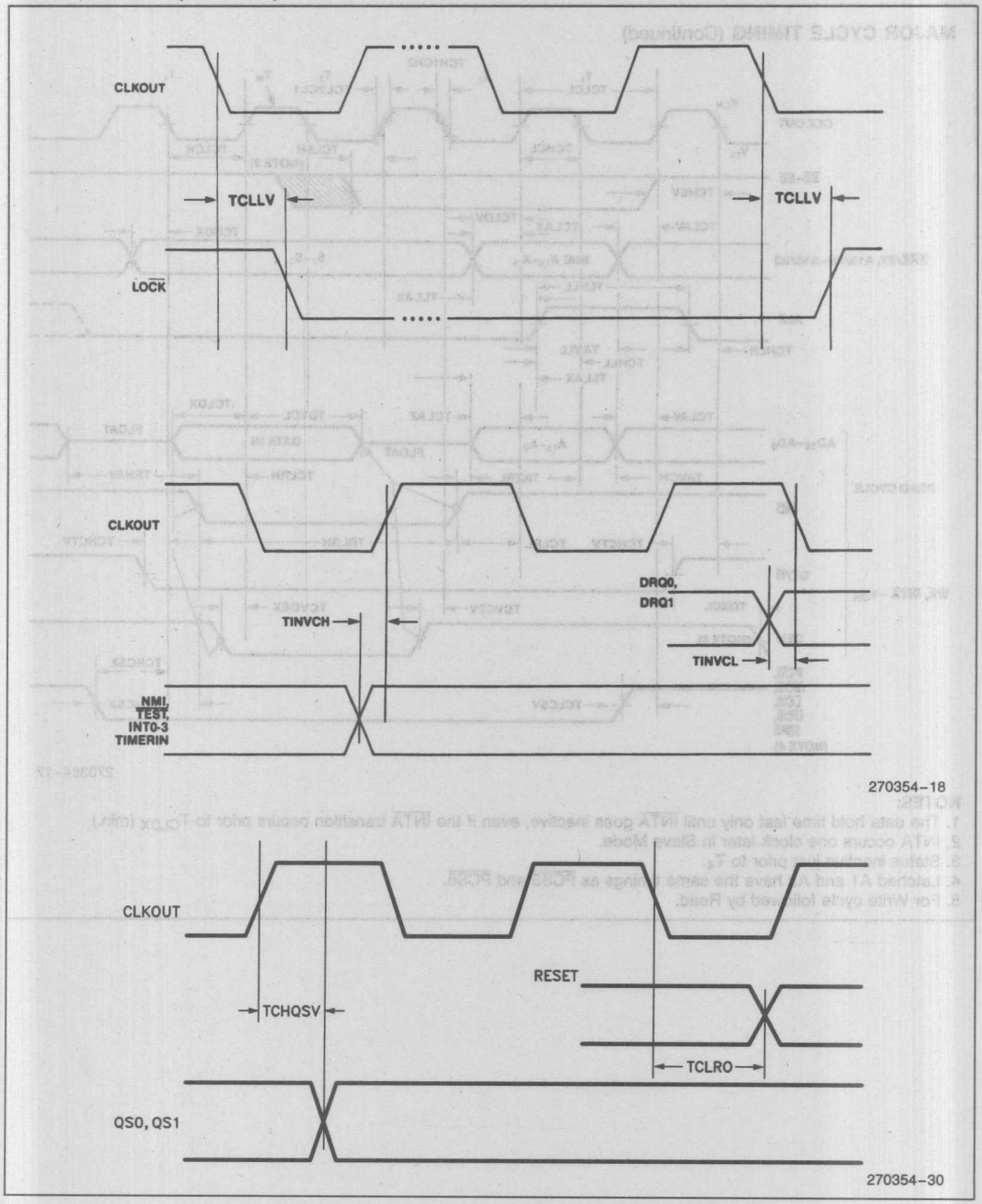
MAJOR CYCLE TIMING (Continued)



270354–17

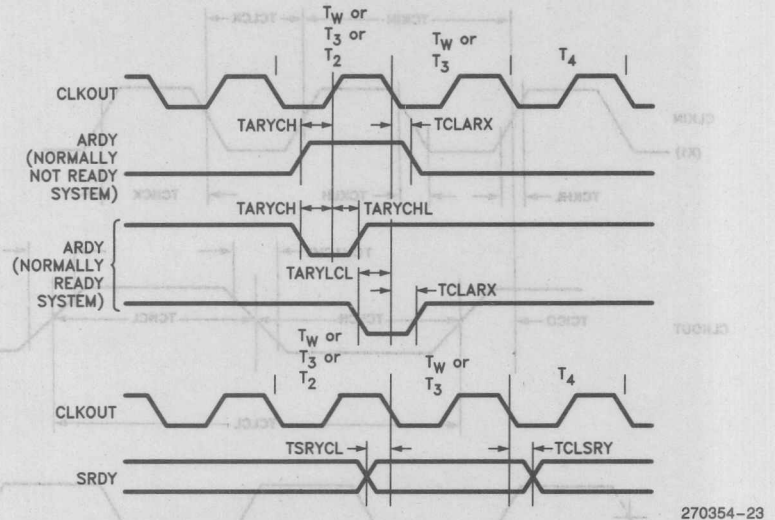
NOTES:
1. The data hold time last only until $\overline{INTA}$ goes inactive, even if the $\overline{INTA}$ transition occurs prior to T_{CLDX} (min.)
2. INTA occurs one clock later in Slave Mode.
3. Status inactive just prior to T_4.
4. Latched A1 and A2 have the same timings as $\overline{PCS5}$ and $\overline{PCS6}$.
5. For Write cycle followed by Read.

WAVEFORMS (Continued)

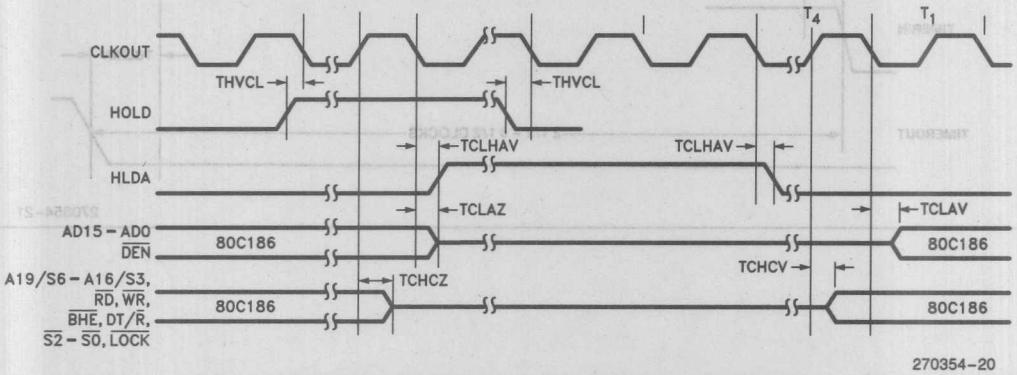


WAVEFORMS (Continued)

READY TIMING

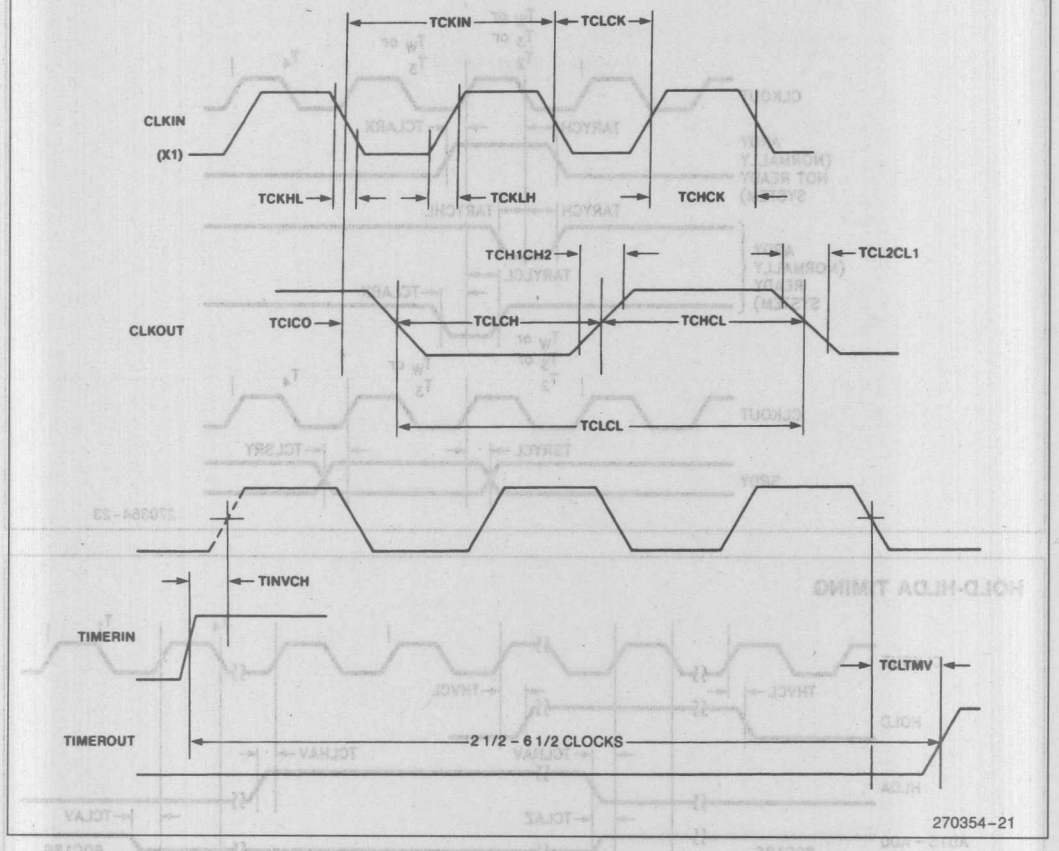


HOLD-HLDA TIMING



WAVEFORMS (Continued)

TIMER ON 80C186



WAVEFORMS (Continued)

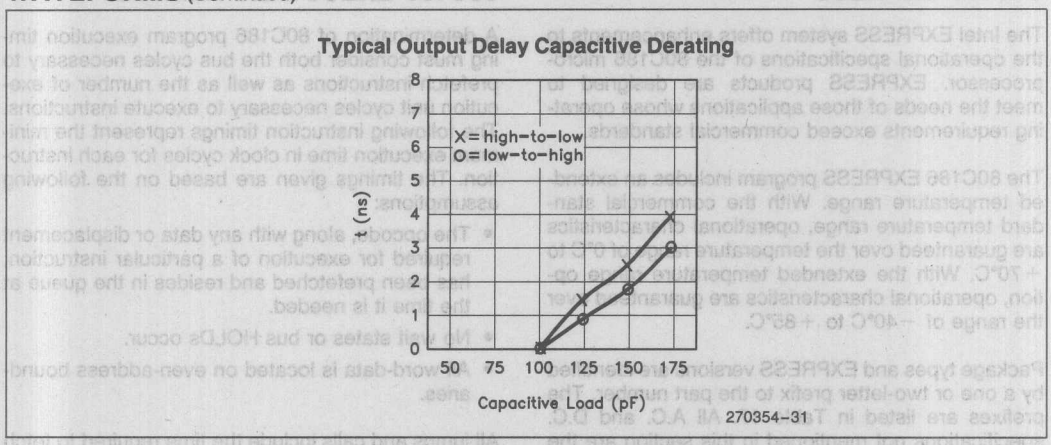


Figure 47. Capacitive Derating Curve

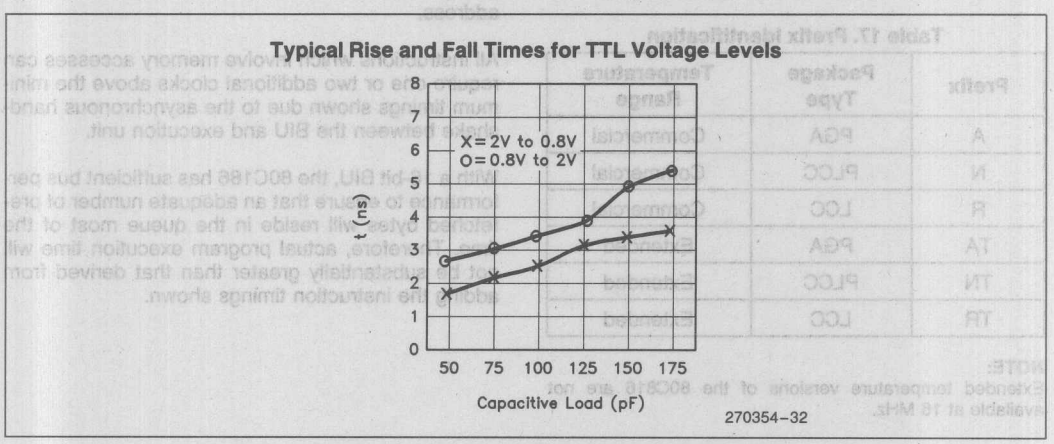


Figure 48. TTL Level Rise and Fall Times for Output Buffers

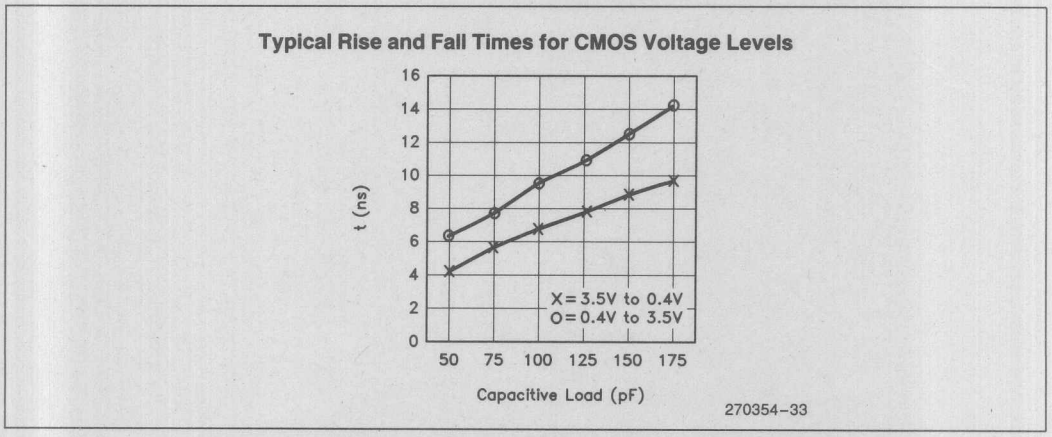


Figure 49. CMOS Level Rise and Fall Times for Output Buffers

80C186 EXPRESS

The Intel EXPRESS system offers enhancements to the operational specifications of the 80C186 microprocessor. EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The 80C186 EXPRESS program includes an extended temperature range. With the commercial standard temperature range, operational characteristics are guaranteed over the temperature range of 0°C to +70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

Package types and EXPRESS versions are identified by a one or two-letter prefix to the part number. The prefixes are listed in Table 17. All A.C. and D.C. specifications not mentioned in this section are the same for both commercial and EXPRESS parts.

Table 17. Prefix Identification

Prefix	Package Type	Temperature Range
A	PGA	Commercial
N	PLCC	Commercial
R	LCC	Commercial
TA	PGA	Extended
TN	PLCC	Extended
TR	LCC	Extended

NOTE:
Extended temperature versions of the 80C816 are not available at 16 MHz.

80C186 EXECUTION TIMINGS

A determination of 80C186 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory accesses can require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

With a 16-bit BIU, the 80C186 has sufficient bus performance to ensure that an adequate number of prefetched bytes will reside in the queue most of the time. Therefore, actual program execution time will not be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1000100w	mod reg r/m			2/12	
Register/memory to register	1000101w	mod reg r/m			2/9	
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1	12–13	8/16-bit
Immediate to register	1011w reg	data	data if w=1		3–4	8/16-bit
Memory to accumulator	1010000w	addr-low	addr-high		8	
Accumulator to memory	1010001w	addr-low	addr-high		9	
Register/memory to segment register	10001110	mod 0 reg r/m			2/9	
Segment register to register/memory	10001100	mod 0 reg r/m			2/11	
PUSH = Push:						
Memory	11111111	mod 1 1 0 r/m			16	
Register	01010 reg				10	
Segment register	000reg110				9	
Immediate	011010s0	data	data if s=0		10	
PUSHA = Push All	01100000				36	
POP = Pop:						
Memory	10001111	mod 0 0 0 r/m			20	
Register	01011 reg				10	
Segment register	000reg111	(reg≠01)			8	
POPA = Pop All	01100001				51	
XCHG = Exchange:						
Register/memory with register	1000011w	mod reg r/m			4/17	
Register with accumulator	10010 reg				3	
IN = Input from:						
Fixed port	1110010w	port			10	
Variable port	1110110w				8	
OUT = Output to:						
Fixed port	1110011w	port			9	
Variable port	1110111w				7	
XLAT = Translate byte to AL	11010111				11	
LEA = Load EA to register	10001101	mod reg r/m			6	
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		18	
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠11)		18	
LAHF = Load AH with flags	10011111				2	
SAHF = Store AH into flags	10011110				3	
PUSHF = Push flags	10011100				9	
POPF = Pop flags	10011101				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	000000dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 000 r/m			3/15	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1111111w	mod 001 r/m			3/15	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10	
Register with register/memory	0011100w	mod reg r/m			3/10	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	
MUL = Multiply (unsigned):	1111011w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
ARITHMETIC (Continued)			
IMUL = Integer multiply (signed):	1111011w | mod101 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43	
IMUL = Integer Immediate multiply (signed)	011010s1 | mod reg r/m | data | data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1111011w | mod110 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44	
IDIV = Integer divide (signed):	1111011w | mod111 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67	
AAM = ASCII adjust for multiply	11010100 | 00001010	19	
AAD = ASCII adjust for divide	11010101 | 00001010	15	
CBW = Convert byte to word	10011000	2	
CWD = Convert word to double word	10011001	4	
LOGIC **Shift/Rotate Instructions:**			
Register/Memory by 1	1101000w | mod TTT r/m	2/15	
Register/Memory by CL	1101001w | mod TTT r/m	5+n/17+n	
Register/Memory by Count	1100000w | mod TTT r/m | count	5+n/17+n	
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR		
AND = **And:**			
Reg/memory and register to either	001000dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod100 r/m | data | data if w=1	4/16	
Immediate to accumulator	0010010w | data | data if w=1	3/4	8/16-bit
TEST = **And function to flags, no result:**			
Register/memory and register	1000010w | mod reg r/m	3/10	
Immediate data and register/memory	1111011w | mod000 r/m | data | data if w=1	4/10	
Immediate data and accumulator	1010100w | data | data if w=1	3/4	8/16-bit
OR = **Or:**			
Reg/memory and register to either	000010dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod001 r/m | data | data if w=1	4/16	
Immediate to accumulator	0000110w | data | data if w=1	3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
LOGIC (Continued)			
XOR = Exclusive or:			
Reg/memory and register to either	001100dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod 110 r/m | data | data if w=1	4/16	
Immediate to accumulator	0011010w | data | data if w=1	3/4	8/16-bit
NOT = Invert register/memory	1111011w | mod 010 r/m	3/10	
STRING MANIPULATION			
MOVS = Move byte/word	1010010w	14	
CMPS = Compare byte/word	1010011w	22	
SCAS = Scan byte/word	1010111w	15	
LODS = Load byte/wd to AL/AX	1010110w	12	
STOS = Store byte/wd from AL/AX	1010101w	10	
INS = Input byte/wd from DX port	0110110w	14	
OUTS = Output byte/wd to DX port	0110111w	14	
Repeated by count in CX (REP/REPE/REPZ/REPNE/REPNZ)			
MOVS = Move string	11110010 | 1010010w	8+8n	
CMPS = Compare string	1111001z | 1010011w	5+22n	
SCAS = Scan string	1111001z | 1010111w	5+15n	
LODS = Load string	11110010 | 1010110w	6+11n	
STOS = Store string	11110010 | 1010101w	6+9n	
INS = Input string	11110010 | 0110110w	8+8n	
OUTS = Output string	11110010 | 0110111w	8+8n	
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	15	
Register/memory indirect within segment	11111111 | mod 010 r/m	13/19	
Direct intersegment	10011010 | segment offset segment selector	23	
Indirect intersegment	11111111 | mod 011 r/m (mod ≠ 11)	38	
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 100 r/m	11/17	
Direct intersegment	11101010 | segment offset segment selector	14	
Indirect intersegment	11111111 | mod 101 r/m (mod ≠ 11)	26	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
RET = **Return from CALL:**						
Within segment	11000011				16	
Within seg adding immed to SP	11000010	data-low	data-high		18	
Intersegment	11001011				22	
Intersegment adding immediate to SP	11001010	data-low	data-high		25	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110 000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = **Interrupt:**						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{TEST}$ = 0
LOCK = Bus lock prefix	11110000	2	
NOP = No Operation	10010000	3	
	(TTT LLL are opcode to processor extension)		

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

REVISION HISTORY

The sections significantly revised since version -002 are:

Block Diagram	Redrawn to illustrate numerics coprocessor interface.
Pin Description Table	Various descriptions rewritten for clarity.
Interrupt Vector Table	Redrawn for clarity. Interrupt Type 16 listed.
ESC Opcode Exception Description	Note added concerning ESC trap.
Oscillator Configurations	Deleted drive of X2 with inverted X1.
RESET Logic	Deleted paragraph concerning setup times for synchronization of multiple processors.
Local Bus Arbitration	Added description of HLDA when a refresh cycle is pending.
Local Bus Controller and Reset	Added description of pullup devices for appropriate pins.
DMA Controller	Added reminder to initialize transfer count registers and pointer registers.
Timers	Added reminder to intialize count registers.
DRAM Refresh Addresses	Refresh address counter described in figure.
D.C. Characteristics	V_{IH2} indicated for SRDY, ARDY. I_{CC} (max.) now indicated for all devices.
Power Supply Current	Typical I_{CC} indicated.
A.C. Characteristics	Input V_{IH} test condition at X1 added. T_{CLDOX}, T_{CVCTV}, T_{CVCTX}, T_{CLHAV}, and T_{CLLV} minimums reduced from 5 ns to 3 ns. T_{CLCH} (min.) and T_{CHCL} (min.) relaxed by 2 ns. Added reminder that T_{SRYCL} and T_{CLSRY} must be met.
Explanation of the A.C. Symbols	New Section.
Major Cycle Timing Waveforms	T_{DXDL} indicated in Read Cycle. T_{CLRO} indicated.
Rise/Fall and Capacitive Derating Curves	New Figures added.
Instruction Set Summary	ESC instruction clock count deleted.

The sections significantly revised since version -001 are:

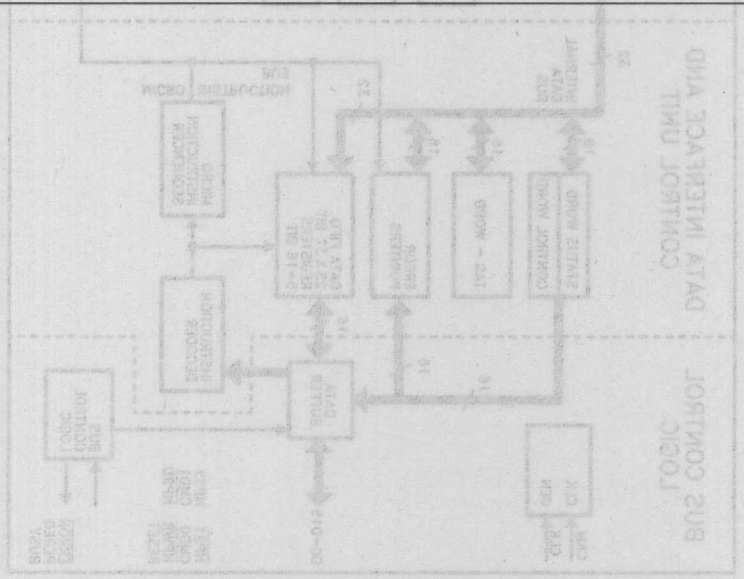
Pin Description Table	Noted $\overline{RES}$ to be low more than 4 clocks.
Oscillator Configurations	Added reminder not to drive X2.
DMA Control Bit Descriptions	Moved and clarified note concerning TC condition for ST/$\overline{STOP}$ clearing during unsynchronized transfers.
Interrupt Controller, etc.	Renamed iRMX Mode to Slave Mode.
Interrupt Request Register	Noted that D0 and D1 are read/write, others read-only.
DRAM Refresh Addresses	Added figure to explain refresh address bits.
A.C. Characteristics	Many timings changed (all listed in ns): T_{CLDX} (min.) from 8 to 5; T_{SRYCL} (min.) from 20 to 5; T_{HVCL} (min.) from 20 to 15; T_{INVCH} (min.) from 25 to 15; T_{INVCL} (min.) from 20 to 15; T_{CLAV} at 12.5 MHz from 4–33 to 5–36; T_{CLAV} at 16 MHz from 4–30 to 5–33; T_{CLAX} (min.) to 0; T_{CLDV} (min.) at 10 MHz from 10 to 5; T_{CLDV} (min.) at 12.5 MHz from 10–33 to 5–36; T_{CLDV} (min.) at 16 MHz from 10–30 to 5–33; T_{CLDOX} (min.) from 10 at 10 MHz and 8 at 12.5 MHz to 5 at both frequencies; T_{CVCTV} (max.) and T_{CHCTV} (max.) at 16 MHz from 25 to 31; T_{CHCTV} (min.) and T_{CVDEX} (min.) both from 10 at 10 MHz and 8 at 12.5 MHz to 5 at both frequencies; T_{CVCTX} (max.) at 16 MHz from 25 to 33; T_{CLRL} at 10 MHz from 10–56 to 5–44; T_{CLRL} at 12.5 MHz from 8–47 to 5–35; T_{CLRL} (max.) at 16 MHz from 25 to 31; T_{CLRH} (min.) at 10 MHz from 10 to 5 and at 12.5 MHz from 8 to 5; T_{CHSV} (min.) at 10 MHz from 10 to 5 and at 12.5 MHz from 8 to 5; T_{CHSV} (max.) at 16 MHz from 25 to 31; T_{CLSH} (min.) at 10 MHz from 10 to 5 and at 12.5 MHz from 8 to 5; T_{CHQSV} (max.) at 12.5 MHz from 23 to 28 and at 16 MHz from 23 to 25; T_{CHDX} (min.) at 10 MHz from 10 to 5 and at 12.5 MHz from 8 to 5; T_{AVCH} (min.) to 0; T_{CLLV} (max.) at 10 MHz from 60 to 45 and at 12.5 MHz from 55 to 40 and at 16 MHz from 40 to 35; T_{DXDL} (min.) to 0; T_{CXCSX} (min.) from 35 at 10 MHz and 29 at 12.5 MHz and 25 at 16 MHz to $T_{CLCH} - 10$ at all frequencies; T_{CHCSX} (min.) at 12.5 MHz and 16 MHz from 4–23 to 5–28 and 5–23 respectively.
Execution Timings	Clarified effect of bus width.

80C187
80-BIT NUMERIC PROCESSOR EXTENSION

- **High Performance 80-Bit Internal Architecture**
- **Two to Three Times 8087 Performance at Equivalent Clock Speed**
- **Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
- **Upward Object-Code Compatible from 8087**
- **Fully Compatible with 80387 and 80387SX. Implements all 80387 Architectural Enhancements over 8087**
- **Directly Interfaces with 80C186 CPU**
- **80C186/80C187 Provide a Software/ Binary Compatible Upgrade from 80186/82188/8087 Systems**
- **Expands 80C186's Data Types to Include 32-, 64-, 80-Bit Floating-Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Directly Extends 80C186's Instruction Set to Trigonometric, Logarithmic, Exponential, and Arithmetic Instructions for All Data Types**
- **Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT, and LOGARITHM**
- **Built-In Exception Handling**
- **Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack**
- **Available in 40-Pin CERDIP and 44-Pin PLCC Package**

(See Packaging Outlines and Dimensions, Order #231369)

The Intel 80C187 is a high-performance numerics processor extension that extends the architecture of the 80C186 with floating-point, extended integer, and BCD data types. A computing system that includes the 80C187 fully conforms to the IEEE Floating-Point Standard. Using a numerics oriented architecture, the 80C187 adds over seventy mnemonics to the instruction set of the 80C186, making a complete solution for high-performance numerics processing. The 80C187 is implemented with 1.5 micron, high-speed CHMOS III technology and packaged in both a 40-pin CERDIP and a 44-pin PLCC package. The 80C187 is upward object-code compatible from the 8087 numerics coprocessor and completely object-code compatible with the 80387 numerics coprocessor.



October 1988
Order Number: 270640-001

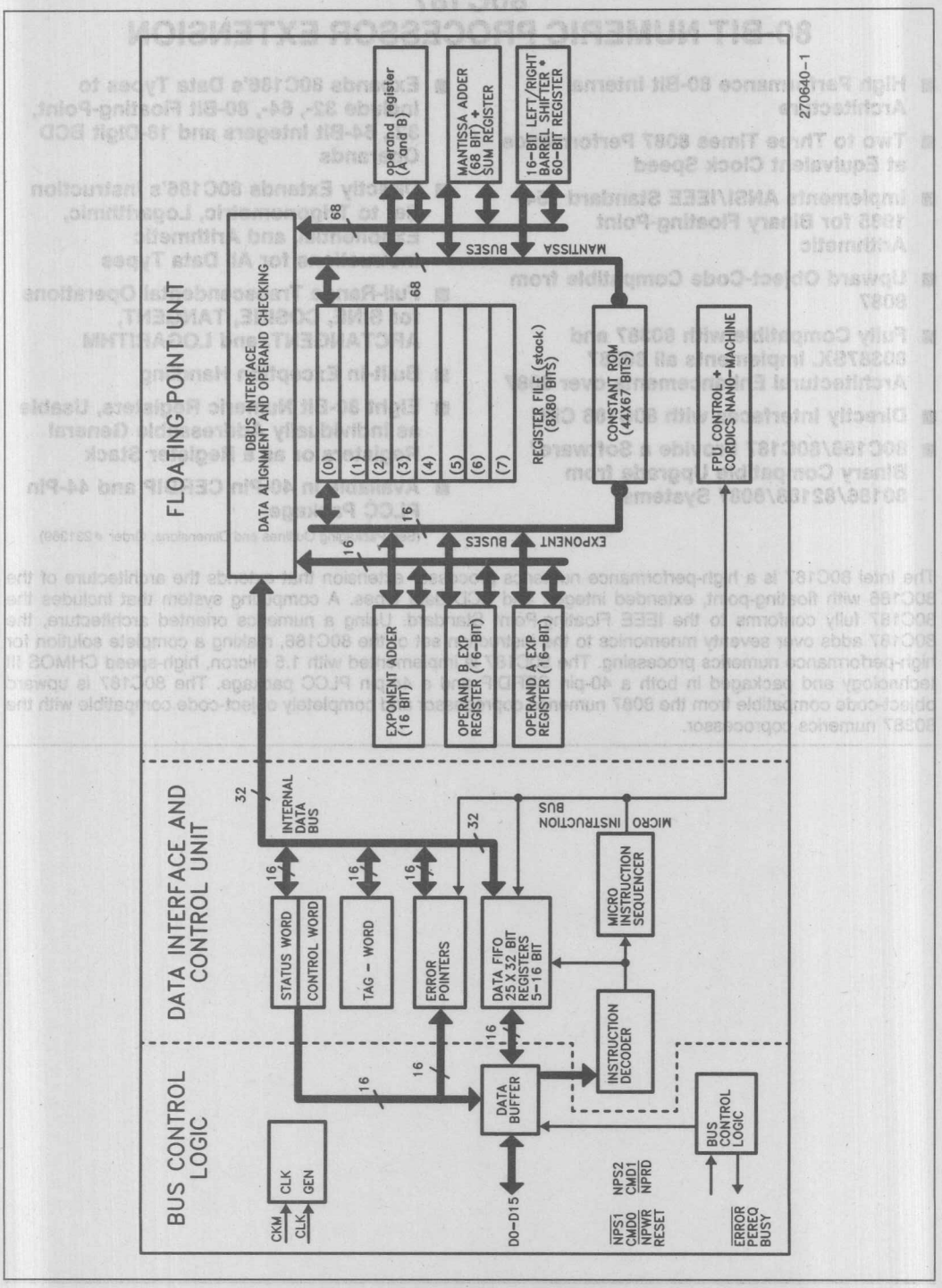


Figure 1. 80C187 Block Diagram

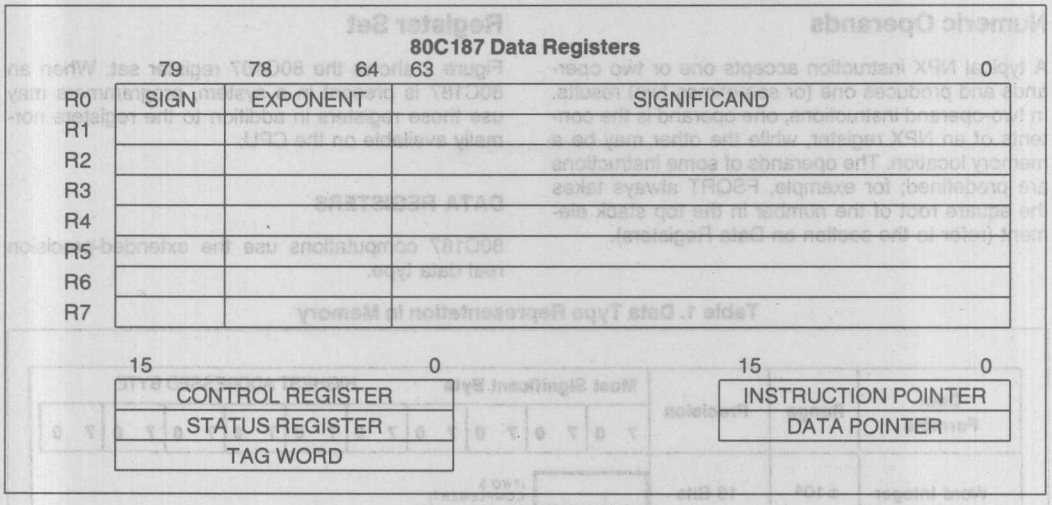


Figure 2. Register Set

FUNCTIONAL DESCRIPTION

The 80C187 Numeric Processor Extension (NPX) provides arithmetic instructions for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The 80C187 effectively extends the register and instruction set of the 80C186 CPU for existing data types and adds several new data types as well. Figure 2 shows the additional registers visible to programs in a system that includes the 80C187. Essentially, the 80C187 can be treated as an additional resource or an extension to the CPU. The 80C186 CPU together with an 80C187 NPX can be used as a single unified system.

A 80C186 system that includes the 80C187 is completely upward compatible with software for the 8086/8087.

The 80C187 interfaces only with the 80C186 CPU. The interface hardware for the 80C187 is not implemented on the 80C188.

PROGRAMMING INTERFACE

The 80C187 adds to the CPU additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the 80C187 requires no special programming tools, because all new instructions and data types are directly supported by the assembler and compilers for high-level languages. The 80C187 supports all 80387 instructions, producing the same binary results.

All communication between the CPU and the 80C187 is transparent to applications software. The CPU automatically controls the 80C187 whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the 80C187. All memory addressing modes are available for addressing numerics operands.

The end of this data sheet lists by class the instructions that the 80C187 adds to the instruction set.

Data Types

Table 1 lists the seven data types that the 80C187 supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at even physical-memory addresses; operands may begin at odd addresses, but will require extra memory cycles to access the entire operand.

Internally, the 80C187 holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

Numeric Operands

A typical NPX instruction accepts one or two operands and produces one (or sometimes two) results. In two-operand instructions, one operand is the contents of an NPX register, while the other may be a memory location. The operands of some instructions are predefined; for example, FSQRT always takes the square root of the number in the top stack element (refer to the section on Data Registers).

Register Set

Figure 2 shows the 80C187 register set. When an 80C187 is present in a system, programmers may use these registers in addition to the registers normally available on the CPU.

DATA REGISTERS

80C187 computations use the extended-precision real data type.

Table 1. Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte ... HIGHEST ADDRESSED BYTE
Word Integer	$\pm 10^4$	16 Bits	(TWO'S COMPLEMENT)
Short Integer	$\pm 10^9$	32 Bits	(TWO'S COMPLEMENT)
Long Integer	$\pm 10^{18}$	64 Bits	(TWO'S COMPLEMENT)
Packed BCD	$\pm 10^{18}$	18 Digits	S, X, MAGNITUDE
Single Precision	$\pm 10^{\pm 38}$	24 Bits	S, BIASED EXPONENT, SIGNIFICAND
Double Precision	$\pm 10^{\pm 308}$	53 Bits	S, BIASED EXPONENT, SIGNIFICAND
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	S, BIASED EXPONENT, I, SIGNIFICAND

270640-2

NOTES:
1. S = Sign bit (0 = Positive, 1 = Negative)
2. d_n = Decimal digit (two per byte)
3. X = Bits have no significance; 80C187 ignores when loading, zeros when storing
4. ▲ = Position of implicit binary point
5. I = Integer bit of significand; stored in temporary real, implicit in single and double precision
6. Exponent Bias (normalized values):
   Single: 127 (7FH)
   Double: 1023 (3FFH)
   Extended Real: 16383 (3FFFH)
7. Packed BCD: $(-1)^S (D_{17} \ldots D_0)$
8. Real: $(-1)^S (2^{E-BIAS}) (F_0, F_1 \ldots)$

The 80C187 register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as individually addressable registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. The 80C187 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit addressing is also relative to TOP.

TAG WORD

The tag word marks the content of each numeric data register, as Figure 3 shows. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the NPX's performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to identify special values (e.g. NaNs or denormals) in the contents of a stack location without the need to perform complex decoding of the actual data.

STATUS WORD

The 16-bit status word (in the status register) shown in Figure 4 reflects the overall state of the 80C187. It may be read and inspected by programs.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It always has the same value as the ES bit (bit 7 of the status word); it does **not** indicate the status of the BUSY output of 80C187.

Bits 13–11 (TOP) point to the 80C187 register that is the current top-of-stack.

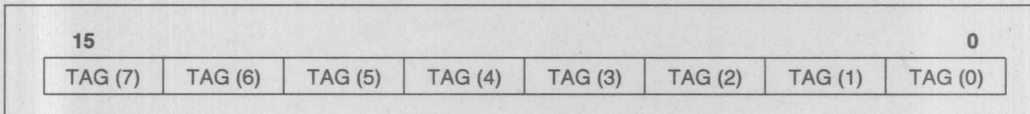
The four numeric condition code bits (C_3–C_0) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2 through 5.

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the $\overline{\text{ERROR}}$ signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 (C_1) distinguishes between stack overflow ($C_1 = 1$) and underflow ($C_1 = 0$).

Figure 4 shows the six exception flags in bits 5–0 of the status word. Bits 5–0 are set to indicate that the 80C187 has detected an exception while executing an instruction. A later section entitled "Exception Handling" explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the $\overline{\text{ERROR}}$ output of the 80C187 is activated immediately.



NOTE:
The index i of tag(i) is **not** top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 3. Tag Word

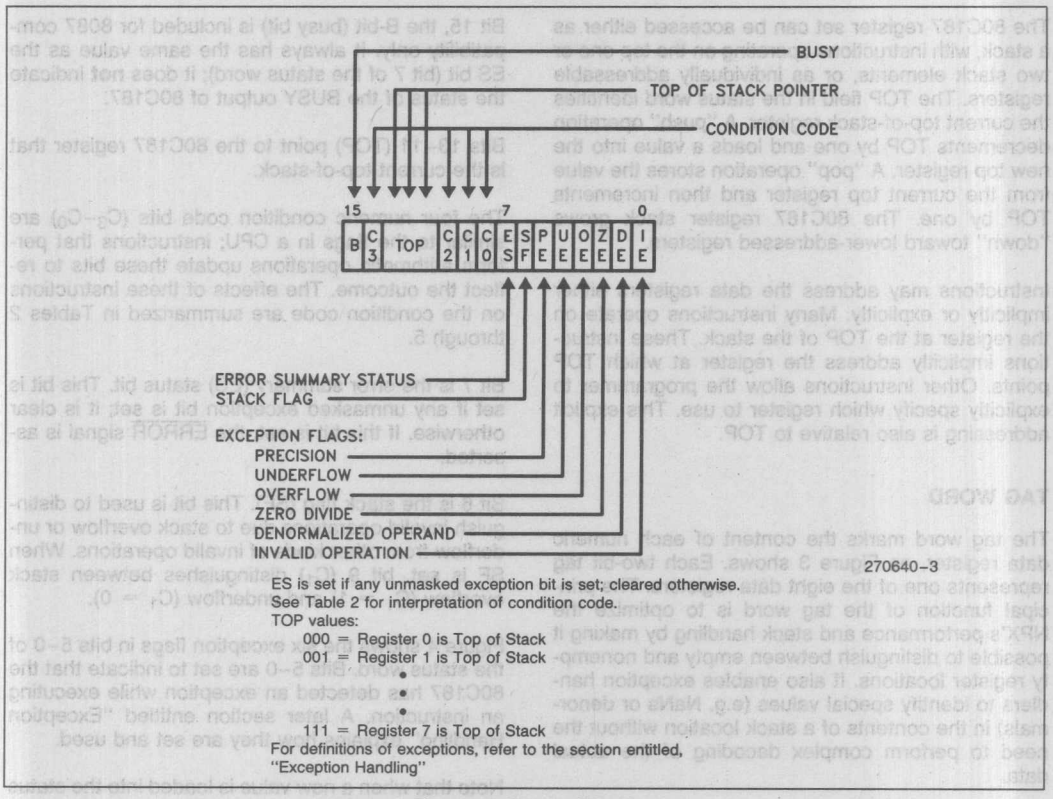


Figure 4. Status Word

CONTROL WORD

The NPX provides several processing options that are selected by loading a control word from memory into the control register. Figure 5 shows the format and encoding of fields in the control word.

Table 2. Condition Code Interpretation

Instruction	C0(S)	C3(Z)	C1(A)	C2(C)
FPREM, FPREM1 (See Table 3)	Three Least Significant Bits of Quotient Q2	Q0	Q1 or O/$\overline{U}$	Reduction 0 = Complete 1 = Incomplete
FCOM, FCOMP, FCOMPP, FTST FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of Comparison (See Table 4)		Zero or O/$\overline{U}$	Operand is not Comparable (Table 4)
FXAM	Operand Class (See Table 5)		Sign or O/$\overline{U}$	Operand Class (Table 5)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant Loads, FXTRACT, FLD, FILD, FBLD, FSTP (Ext Real)	UNDEFINED		Zero or O/$\overline{U}$	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/$\overline{U}$	UNDEFINED
FPTAN, FSIN, FCOS, FSINCOS	UNDEFINED		Roundup or O/$\overline{U}$, Undefined if C2 = 1	Reduction 0 = Complete 1 = Incomplete
FLDENV, FRSTOR	Each Bit Loaded from Memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/$\overline{U}$ When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Reduction If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.

Roundup When the PE bit of the status word is set, this bit indicates whether one was added to the least significant bit of the result during the last rounding.

UNDEFINED Do not rely on finding any specific value in these bits.

The low-order byte of this control word configures exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 80C187 recognizes.

The high-order byte of the control word configures the 80C187 operating mode, including precision, rounding, and infinity control.

- The "infinity control bit" (bit 12) is not meaningful to the 80C187, and programs must ignore its value. To maintain compatibility with the 8087, this bit can be programmed; however, regardless of its value, the 80C187 always treats infinity in the affine sense ($-\infty < +\infty$). This bit is initialized to zero both after a hardware reset and after the FINIT instruction.
- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.
- The precision control (PC) bits (bits 9–8) can be used to set the 80C187 internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

Table 3. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1
C2	C3	C1	C0	
1	X	X	X	Incomplete Reduction: Further Iteration Required for Complete Reduction
0	Q1	Q0	Q2	Q MOD 8
	0	0	0	0
	0	1	0	1
	1	0	0	2
	1	1	0	3
	0	0	1	4
	0	1	1	5
	1	0	1	6
	1	1	1	7

Complete Reduction: C0, C3, C1 Contain Three Least Significant Bits of Quotient

Table 4. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 5. Condition Code Defining Operand Class

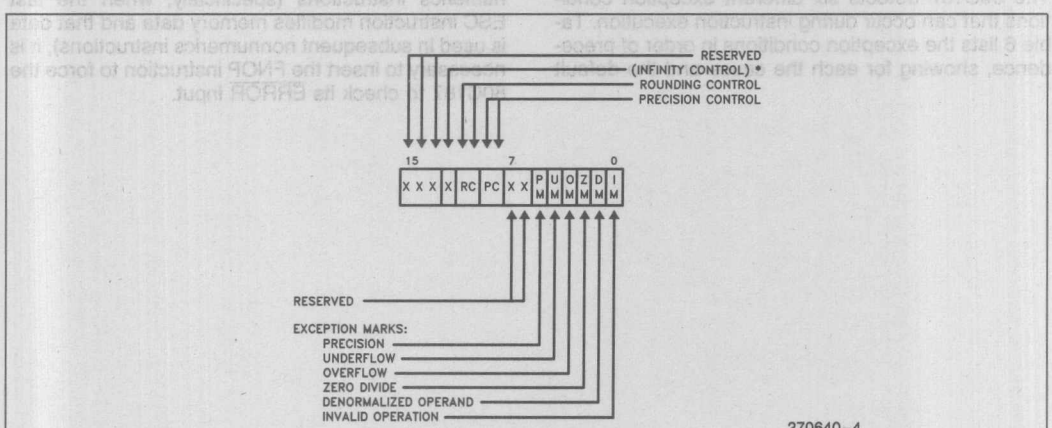
C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	1	− Denormal

INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any exceptions detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numerics instruction, the 80C187 contains registers that aid in diagnosis. These registers supply the opcode of the failing numerics instruction, the address of the instruction, and the address of its numerics memory operand (if appropriate).

The instruction and data pointers are provided for user-written exception handlers. Whenever the 80C187 executes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in the format shown by Figure 6. The ESC instruction FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values between the registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.



Precision Control
00— 24 Bits (Single Precision)
01— (Reserved)
10— 53 Bits (Double Precision)
11— 64 Bits (Extended Precision)

Rounding Control
00— Round to Nearest or Even
01— Round Down (toward $-\infty$)
10— Round Up (toward $+\infty$)
11— Chop (Truncate toward Zero)

*The "infinity control" bit is not meaningful to the 80C187. To maintain compatibility with the 8087, this bit can be programmed; however, regardless of its value, the 80C187 treats infinity in the affine sense ($-\infty < +\infty$).

Figure 5. Control Word

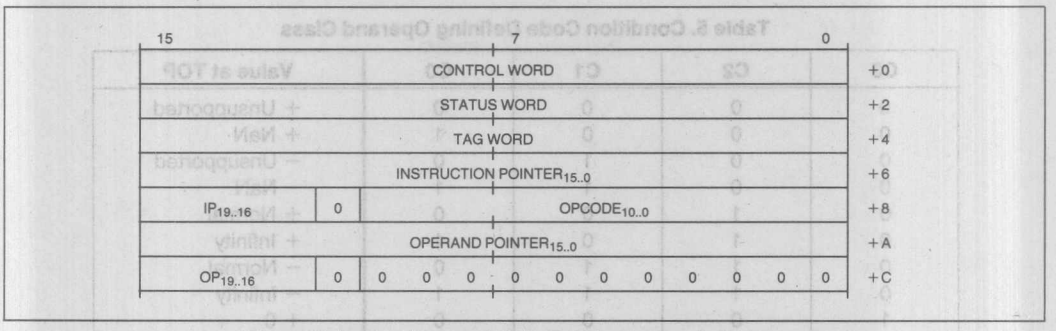


Figure 6. Instruction and Data Pointer Image in Memory

Interrupt Description

CPU interrupt 16 is used to report exceptional conditions while executing numeric programs. Interrupt 16 indicates that the previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC instructions can cause this interrupt. The CPU return address pushed onto the stack of the exception handler points to an ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the NPX. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

Exception Handling

The 80C187 detects six different exception conditions that can occur during instruction execution. Table 6 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the 80C187 if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the $\overline{\text{ERROR}}$ signal. When the CPU attempts to execute another ESC instruction, interrupt 16 occurs. The exception condition must be resolved via an interrupt service routine. The return address pushed onto the CPU stack upon entry to the service routine does not necessarily point to the failing instruction nor to the following instruction. The 80C187 saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

If error trapping is required at the end of a series of numerics instructions (specifically, when the last ESC instruction modifies memory data and that data is used in subsequent nonnumerics instructions), it is necessary to insert the FNOP instruction to force the 80C187 to check its $\overline{\text{ERROR}}$ input.

Table 6. Exceptions

Exception	Cause	Default Action (If Exception is Masked)
Invalid Operation	Operation on a signalling NaN, unsupported format, indeterminate form (0* ∞, 0/0), (+ ∞) + (− ∞), etc.), or stack overflow/underflow (SF is also set)	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e. it has the smallest exponent but a nonzero significand	The operand is normalized, and normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode	Normal processing continues

Initialization

After FNINIT or RESET, the control word contains the value 037FH (all exceptions masked, precision control 64 bits, rounding to nearest) the same values as in an 8087 after RESET. For compatibility with the 8087, the bit that used to indicate infinity control (bit 12) is set to zero; however, regardless of its setting, infinity is treated in the affine sense. After FNINIT or RESET, the status word is initialized as follows:

- All exceptions are set to zero.
- Stack TOP is zero, so that after the first push the stack top will be register seven (111B).
- The condition code C_3–C_0 is **undefined**.
- The B-bit is zero.

The tag word contains FFFFH (all stack locations are empty).

80C186/80C187 initialization software should execute an FNINIT instruction (i.e. an FINIT without a preceding WAIT) after RESET. The FNINIT is not strictly required for 80C187 software, but Intel recommends its use to help ensure upward compatibility with other processors.

8087 Compatibility

This section summarizes the differences between the 80C187 and the 8087. Many changes have been designed into the 80C187 to directly support the IEEE standard in hardware. These changes result in increased performance by elminating the need for software that supports the standard.

GENERAL DIFFERENCES

The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80C187 Numeric Processor Extension. They do not alter the state of the 80C187 Numeric Processor Extension. (They are treated similarly to FNOP, except that ERROR is not checked.) While 8086/8087 code containing these instructions can be executed on the 80C186/80C187, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80C187 Numeric Processor Extension.

The 80C187 differs from the 8087 with respect to instruction, data, and exception synchronization. Except for the processor control instructions, all of the 80C187 numeric instructions are automatically synchronized by the 80C186 CPU. When necessary, the

80C186 automatically tests the BUSY line from the 80C187 Numeric Processor Extension to ensure that the 80C187 Numeric Processor Extension has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 CPUs, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute on the 80C186/80C187, these WAIT instructions are unnecessary.

The 80C187 supports only affine closure for infinity arithmetic, not projective closure.

Operands for FSCALE and FPATAN are no longer restricted in range (except for $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.

Rounding control is in effect for FLD *constant*.

Software cannot change entries of the tag word to values (other than empty) that differ from actual register contents.

After reset, FINIT, and incomplete FPREM, the 80C187 resets to zero the condition code bits C_3–C_0 of the status word.

In conformance with the IEEE standard, the 80C187 does not support the special data formats pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

The denormal exception has a different purpose on the 80C187. A system that uses the denormal-exception handler solely to normalize the denormal operands, would better mask the denormal exception on the 80C187. The 80C187 automatically normalizes denormal operands when the denormal exception is masked.

EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the 80C186/80C187:

1. The 80C186/80C187 traps exceptions only on the next ESC instruction; i.e. the 80C186 does not notice unmasked 80C187 exceptions on the 80C186 $\overline{\text{ERROR}}$ input line until a later numerics instruction is executed. To force the 80C186 to sample its $\overline{\text{ERROR}}$ input, existing high-level compilers and assembly-language programmers typically insert WAIT and FWAIT for this purpose. Because the 80C186 does not sample $\overline{\text{ERROR}}$ on WAIT and FWAIT instructions, programmers should place an FNOP instruction at the end of a sequence of numerics instructions to force the 80C187 to sample its $\overline{\text{ERROR}}$ input.
2. The 80C187 Numeric Processor Extension signals exceptions through a dedicated $\overline{\text{ERROR}}$ line to the CPU. The 80C187 error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numerics exception handlers for the 8086/8087 should be deleted.
3. Interrupt vector 16 must point to the numerics exception handling routine.
4. The ESC instruction address saved in the 80C187 Numeric Processor Extension includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. When the overflow or underflow exception is masked, the 80C187 differs from the 8087 in rounding when overflow or underflow occurs. The 80C187 produces results that are consistent with the rounding mode.
6. When the underflow exception is masked, the 80C187 sets its underflow flag only if there is also a loss of accuracy during denormalization.
7. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
8. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
9. The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
10. The denormal exception no longer takes precedence over all other exceptions.
11. When the denormal exception is masked, the 80C187 automatically normalizes denormal operands. The 8087 performs unnormal arithmetic, which might produce an unnormal result.
12. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
13. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
14. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
15. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized oper-

and exception. When loading a signalling NaN, FLD *single/double precision* signals an invalid-operand exception.

16. The 80C187 only generates quiet NaNs (as on the 8087); however, the 80C187 distinguishes between quiet NaNs and signalling NaNs. Signalling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
17. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 8087 leaves the original operand in ST(1) intact.
18. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
    - FSCALE (0, ∞) generates the invalid operation exception.
    - FSCALE (finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE (finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

    The 8087 returns zero in the first case and raises the invalid-operation exception in the other cases.
19. The 80C187 returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 supports a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

Table 7. Pin Summary

Pin Name	Function	Active State	Input/ Output
CLK	CLocK		I
CKM	ClockIng Mode		I
RESET	System reset	High	I
PEREQ	Processor Extension REQuest	High	O
BUSY	Busy status	High	O
$\overline{\text{ERROR}}$	Error status	Low	O
D_{15}–D_0	Data pins	High	I/O
$\overline{\text{NPRD}}$	Numeric Processor ReaD	Low	I
$\overline{\text{NPWR}}$	Numeric Processor WRite	Low	I
$\overline{\text{NPS1}}$	NPX select #1	Low	I
NPS2	NPX select #2	High	I
CMD0	CoMmanD 0	High	I
CMD1	CoMmanD 1	High	I
V_{CC}	System power		I
V_{SS}	System ground		I

HARDWARE INTERFACE

In the following description of hardware interface, an overbar above a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no overbar is present above the signal name, the signal is asserted when at the high voltage level.

Signal Description

In the following signal descriptions, the 80C187 pins are grouped by function as follows:

1. Execution Control— CLK, CKM, RESET
2. NPX Handshake— PEREQ, BUSY, $\overline{\text{ERROR}}$
3. Bus Interface Pins— D_{15}–D_0, $\overline{\text{NPWR}}$, $\overline{\text{NPRD}}$
4. Chip/Port Select— $\overline{\text{NPS1}}$, NPS2, CMD0, CMD1
5. Power Supplies— V_{CC}, V_{SS}

Table 7 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. Figure 7 shows the locations of pins on the CERDIP package, while Figure 8 shows the locations of pins on the PLCC package. Table 8 helps to locate pin identifiers in Figures 7 and 8.

Clock (CLK)

This input provides the basic timing for internal operation. This pin does not require MOS-level input; it will operate at either TTL or MOS levels up to the maximum allowed frequency. A minimum frequency must be provided to keep the internal logic properly functioning. Depending on the signal on CKM, the signal on CLK can be divided by two to produce the internal clock signal (in which case CLK may be up to 32 MHz in frequency), or can be used directly (in which case CLK may be up to 12.5 MHz).

Clocking Mode (CKM)

This pin is a strapping option. When it is strapped to V_{CC} (HIGH), the CLK input is used directly; when strapped to V_{SS} (LOW), the CLK input is divided by two to produce the internal clock signal. During the RESET sequence, this input must be stable at least four internal clock cycles (i.e. CLK clocks when CKM is HIGH; 2 × CLK clocks when CKM is LOW) before RESET goes LOW.

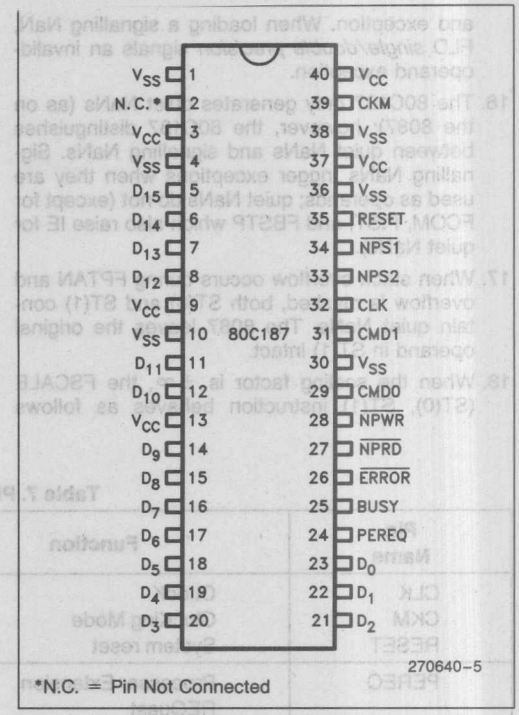


Figure 7. CERDIP Pin Configuration

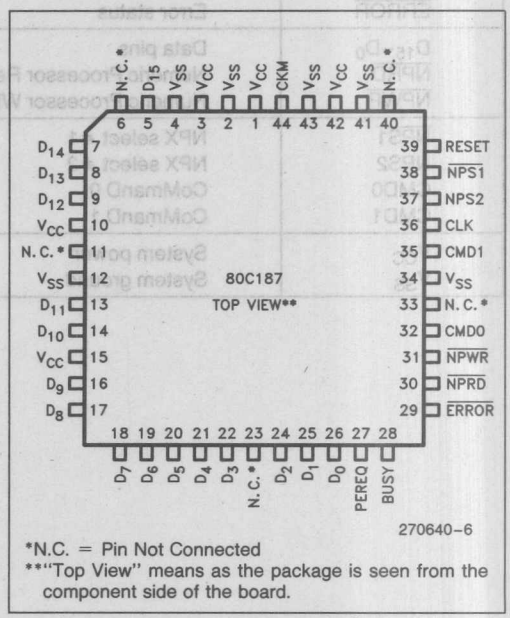


Figure 8. PLCC Pin Configuration

Table 8. PLCC Pin Cross-Reference

Pin Name	CERDIP Package	PLCC Package
BUSY	25	28
CKM	39	44
CLK	32	36
CMD0	29	32
CMD1	31	35
D_0	23	26
D_1	22	25
D_2	21	24
D_3	20	22
D_4	19	21
D_5	18	20
D_6	17	19
D_7	16	18
D_8	15	17
D_9	14	16
D_{10}	12	14
D_{11}	11	13
D_{12}	8	9
D_{13}	7	8
D_{14}	6	7
D_{15}	5	5
$\overline{ERROR}$	26	29
No Connect	2	6, 11, 23, 33, 40
$\overline{NPRD}$	27	30
$\overline{NPS1}$	34	38
NPS2	33	37
$\overline{NPWR}$	28	31
PEREQ	24	27
RESET	35	39
V_{CC}	3, 9, 13, 37, 40	1, 3, 10, 15, 42
V_{SS}	1, 4, 10, 30, 36, 38	2, 4, 12, 34, 41, 43

System Reset (RESET)

A LOW to HIGH transition on this pin causes the 8C187 to terminate its present activity and to enter a dormant state. RESET must remain active (HIGH) for at least four internal clock periods. (The relation of the internal clock period to CLK depends on CLKM; the internal clock may be different from that of the CPU.) Note that the 80C187 is active internally for 25 clock periods after the termination of the RESET signal (the HIGH to LOW transition of RESET); therefore, the first instruction should not be written to the 80C187 until 25 internal clocks after the falling edge of RESET. Table 9 shows the status of the output pins during the reset sequence. After a reset, all output pins return to their inactive states.

Table 9. Output Pin Status during Reset

Output Pin Name	Value during Reset
BUSY	HIGH
$\overline{ERROR}$	HIGH
PEREQ	LOW
D_{15}–D_0	TRI-STATE OFF

Processor Extension Request (PEREQ)

When active, this pin signals to the CPU that the 80C187 is ready for data transfer to/from its data FIFO. When there are more than five data transfers,

PEREQ is deactivated after the first three transfers and subsequently after every four transfers. This signal always goes inactive before BUSY goes inactive.

Busy Status (BUSY)

When active, this pin signals to the CPU that the 80C187 is currently executing an instruction. This pin is active HIGH. It should be connected to the 80C186's TEST/BUSY pin. During the RESET sequence this pin is HIGH. The 80C186 uses this HIGH state to detect the presence of an 80C187.

Error Status ($\overline{\text{ERROR}}$)

This pin reflects the ES bit of the status register. When active, it indicates that an unmasked exception has occurred. This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, FNSAVE, FLDCW, FLDENV, and FRSTOR. This pin should be connected to the $\overline{\text{ERROR}}$ pin of the CPU. $\overline{\text{ERROR}}$ can change state only when BUSY is active.

Data Pins (D_{15}–D_0)

These bidirectional pins are used to transfer data and opcodes between the CPU and 80C187. They are normally connected directly to the corresponding CPU data pins. Other buffers/drivers driving the local data bus must be disabled when the CPU reads from the NPX. High state indicates a value of one. D_0 is the least significant data bit.

Numeric Processor Write ($\overline{\text{NPWR}}$)

A signal on this pin enables transfers of data from the CPU to the NPX. This input is valid only when $\overline{\text{NPS1}}$ and NPS2 are both active.

Numeric Processor Read ($\overline{\text{NPRD}}$)

A signal on this pin enables transfers of data from the NPX to the CPU. This input is valid only when $\overline{\text{NPS1}}$ and NPS2 are both active.

Numeric Processor Selects ($\overline{\text{NPS1}}$ and NPS2)

Concurrent assertion of these signals indicates that the CPU is performing an escape instruction and enables the 80C187 to execute that instruction. No data transfer involving the 80C187 occurs unless the device is selected by these lines.

Command Selects (CMD0 and CMD1)

These pins along with the select pins allow the CPU to direct the operation of the 80C187.

System Power (V_{CC})

System power provides the +5V ±10% DC supply input. All V_{CC} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

System Ground (V_{SS})

All V_{SS} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

Processor Architecture

As shown by the block diagram (Figure 1), the 80C187 NPX is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating-point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numerics instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for CPU bus tracking and interface.

BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the 80C187 and transferring outputs from the 80C187 to memory. A dedicated communication protocol makes possible high-speed transfer of opcodes and operands between the CPU and 80C187.

Table 10. Bus Cycles Definition

$\overline{NPS1}$	NPS2	CMD0	CMD1	$\overline{NPRD}$	$\overline{NPWR}$	Bus Cycle Type
x	0	x	x	x	x	80C187 Not Selected
1	x	x	x	x	x	80C187 Not Selected
0	1	0	0	1	0	Opcode Write to 80C187
0	1	0	0	0	1	CW or SW Read from 80C187
0	1	1	0	0	1	Read Data from 80C187
0	1	1	0	1	0	Write Data to 80C187
0	1	0	1	1	0	Write Exception Pointers
0	1	0	1	0	1	Reserved
0	1	1	1	0	1	Read Opcode Status
0	1	1	1	1	0	Reserved

DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, FSTCW, FSETPM, or FRSTPM, the control executes it independently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY, PEREQ, and $\overline{ERROR}$ signals that synchronize 80C187 activities with the CPU.

FLOATING-POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

Bus Cycles

The pins $\overline{NPS1}$, NPS2, CMD0, CMD1, $\overline{NPRD}$ and $\overline{NPWR}$ identify bus cycles for the NPX. Table 10 defines the types of 80C187 bus cycles.

80C187 ADDRESSING

The $\overline{NPS1}$, NPS2, CMD0, and CMD1 signals allow the NPX to identify which bus cycles are intended for the NPX. The NPX responds to I/O cycles when the I/O address is 00F8H, 00FAH, 00FCH, or 00FEH. The correspondence betwen I/O addresses and control signals is defined by Table 11. To guarantee correct operation of the NPX, programs must not perform any I/O operations to these reserved port addresses.

Table 11. I/O Address Decoding

I/O Address (Hexadecimal)	80C187 Select and Command Inputs			
	NPS2	$\overline{NPS1}$	CMD1	CMD0
00F8	1	0	0	0
00FA	1	0	0	1
00FC	1	0	1	0
00FE	1	0	1	1

CPU/NPX SYNCHRONIZATION

The pins BUSY, PEREQ, and $\overline{\text{ERROR}}$ are used for various aspects of synchronization between the CPU and the NPX.

BUSY is used to synchronize instruction transfer from the CPU to the 80C187. When the 80C187 recognizes an ESC instruction, it asserts BUSY. For most ESC instructions, the CPU waits for the 80C187 to deassert BUSY before sending the new opcode.

The NPX uses the PEREQ pin of the CPU to signal that the NPX is ready for data transfer to or from its data FIFO. The NPX does not directly access memory; rather, the CPU provides memory access services for the NPX.

Once the CPU initiates an 80C187 instruction that has operands, the CPU waits for PEREQ signals that indicate when the 80C187 is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the CPU continues program execution while the 80C187 executes the ESC instruction.

In 8086/8087 systems, WAIT instructions are required to achieve synchronization of both commands and operands. The 80C187, however, does not require WAIT instructions. The WAIT or FWAIT instruction commonly inserted by high-level compilers and assembly-language programmers for exception synchronization is not treated as an instruction by the 80C186 and does not provide exception trapping. (Refer to the section "System Configuration for 8087-Compatible Exception Trapping".)

Once it has started to execute a numerics instruction and has transferred the operands from the CPU, the 80C187 can process the instruction in parallel with and independent of the host CPU. When the NPX detects an exception, it asserts the $\overline{\text{ERROR}}$ signal, which causes a CPU interrupt.

OPCODE INTERPRETATION

The CPU and the NPX use a bus protocol that adapts to the numerics opcode being executed. Only the NPX directly interprets the opcode. Some of the results of this interpretation are relevant to the CPU. The NPX records these results (opcode status information) in an internal 16-bit register. The 80C186 accesses this register only via reads from NPX port 00FEH. Tables 10 and 11 define the signal combinations that correspond to each of the following steps.

1. The CPU writes the opcode to NPX port 00F8H. This write can occur even when the NPX is busy or is signalling an exception. The NPX does not necessarily begin executing the opcode immediately.
2. The CPU reads the opcode status information from NPX port 00FEH.
3. The CPU initiates subsequent bus cycles according to the opcode status information. The opcode status information specifies whether to wait until the NPX is not busy, when to transfer exception pointers to port 00FCH, when to read or write operands and results at port 00FAH, etc.

For most instructions, the NPX does not start executing the previously transferred opcode until the CPU (guided by the opcode status information) first writes exception pointer information to port 00FCH of the NPX. This protocol is completely transparent to programmers.

Bus Operation

With respect to bus interface, the 80C187 is fully asynchronous with the CPU, even when it operates from the same clock source as the CPU. The CPU initiates a bus cycle for the NPX by activating both $\overline{\text{NPS1}}$ and NPS2, the NPX select signals. During the CLK period in which $\overline{\text{NPS1}}$ and NPS2 are activated, the 80C187 also examines the $\overline{\text{NPRD}}$ and $\overline{\text{NPRW}}$

input signals to determine whether the cycle is a read or a write cycle and examines the CMD0 and CMD1 inputs to determine whether an opcode, operand, or control/status register transfer is to occur. The 80C187 activates its BUSY output some time after the leading edge of the $\overline{NPRD}$ or $\overline{NPRW}$ signal. Input and ouput data are referenced to the trailing edges of the $\overline{NPRD}$ and $\overline{NPRW}$ signals.

The 80C187 activates the PEREQ signal when it is ready for data transfer. The 80C187 deactivates PEREQ automatically.

System Configuration

The 80C187 can be connected to the 80C186 CPU as shown by Figure 9. (Refer to the 80C186 Data Sheet for an explanation of the 80C186's signals.) This interface has the following characteristics:

- The 80C187's $\overline{NPS1}$, $\overline{ERROR}$, PEREQ, and BUSY pins are connected directly to the corresponding pins of the 80C186.
- The 80C186 pin $\overline{MCS3}/\overline{NPS}$ is connected to $\overline{NPS1}$; NPS2 is connected to V_{CC}.
- The $\overline{NPRD}$ and $\overline{NPRW}$ pins are connected to the $\overline{RD}$ and $\overline{WR}$ pins of the 80C186.
- CMD1 and CMD0 come from the latched A_2 and A_1 of the 80C186, respectively.
- The 80C187 BUSY output connects to the 80C186 $\overline{TEST}$/BUSY input. During RESET, the signal at the 80C187 BUSY output automatically programs the 80C186 to use the 80C187.
- The 80C187 can use the CLKOUT signal of the 80C186 to conserve board space when operating at 12.5 MHz or less. In this case, the 80C187 CKM input must be pulled HIGH. For operation in excess of 12.5 MHz, a double-frequency external oscillator for CLK input is needed. In this case, CKM must be pulled LOW.

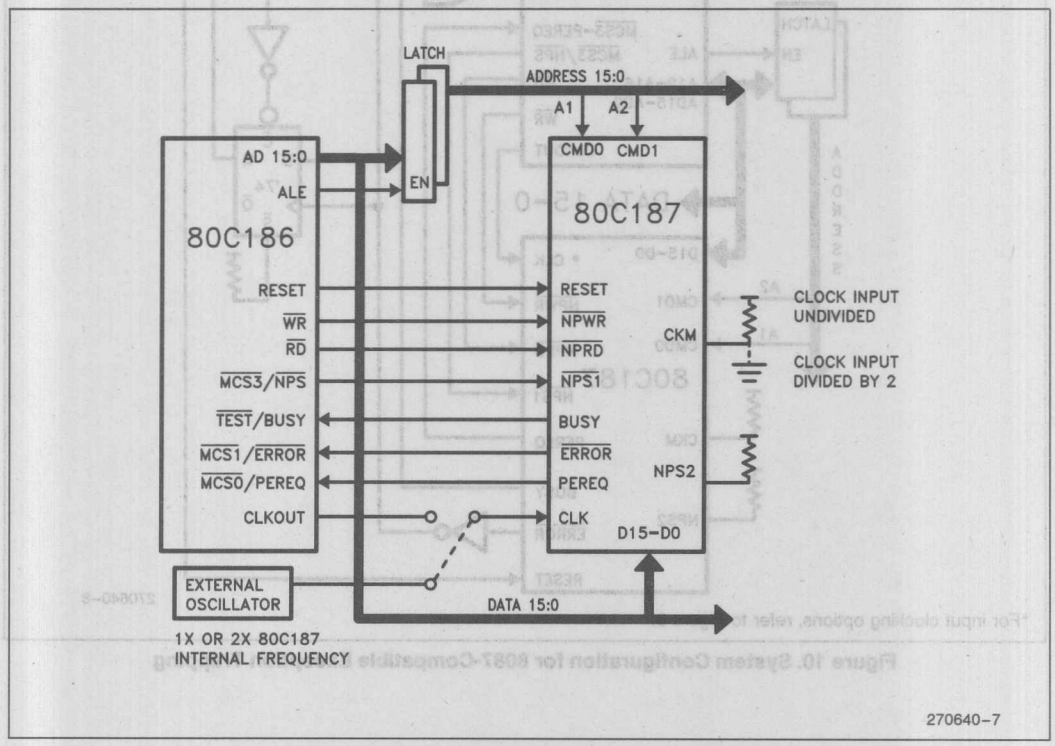
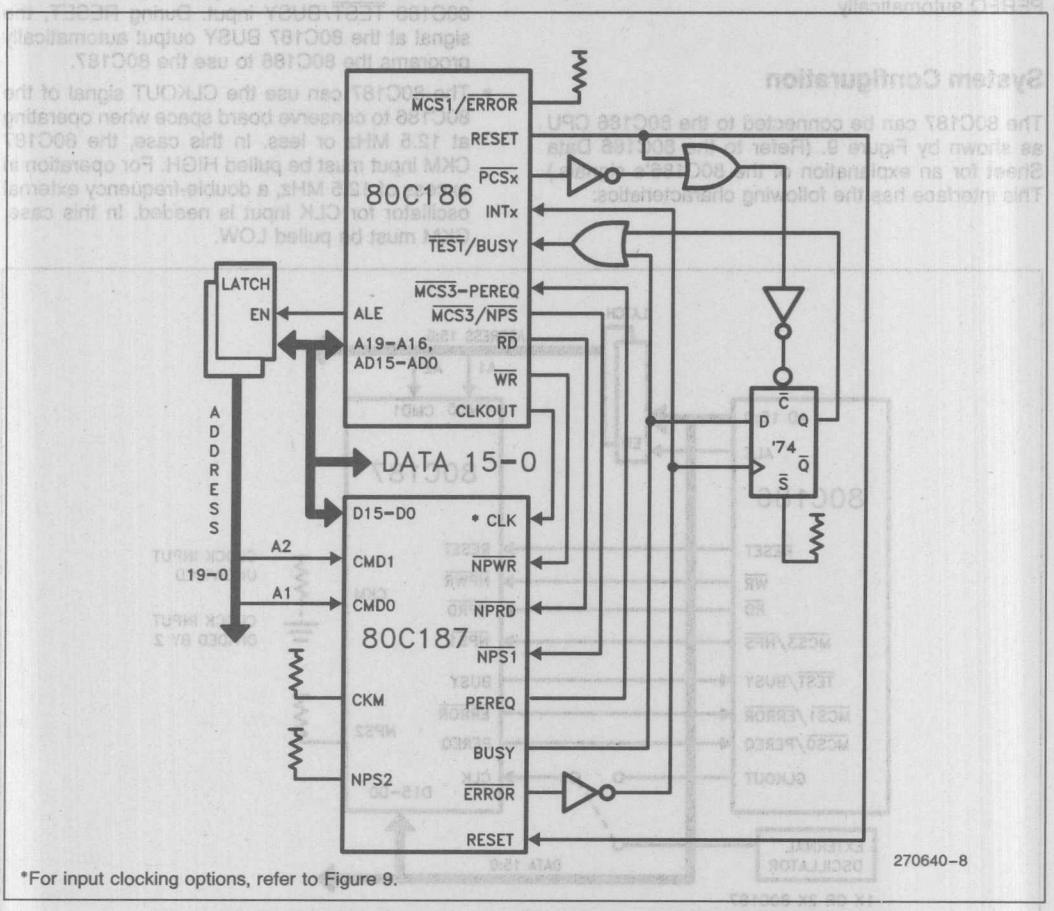


Figure 9. 80C186/80C187 System Configuration

System Configuration for 80186/ 80187-Compatible Exception Trapping

When the 80C187 $\overline{\text{ERROR}}$ output signal is connected directly to the 80C186 $\overline{\text{ERROR}}$ input, floating-point exceptions cause interrupt #16. However, existing software may be programmed to expect floating-point exceptions to be signalled over an external interrupt pin via an interrupt controller.

For exception handling compatible with the 80186/ 82188/8087, the 80C186 can be wired to recognize exceptions through an external interrupt pin, as Figure 10 shows. (Refer to the 80C186 Data Sheet for an explanation of the 80C186's signals.) With this arrangement, a flip-flop is needed to latch BUSY upon assertion of $\overline{\text{ERROR}}$. The latch can then be cleared during the exception-handler routine by forcing a $\overline{\text{PCS}}$ pin active. The latch must also be cleared at RESET in order for the 80C186 to work with the 80C187.



*For input clocking options, refer to Figure 9.

Figure 10. System Configuration for 8087-Compatible Exception Trapping

ELECTRICAL DATA

Absolute Maximum Ratings*

Case Temperature Under Bias (T_C) . . . 0°C to +85°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin
with Respect to Ground . . . . −0.5V to V_{CC} +0.5V
Power Dissipation 1.5W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

Power and Frequency Requirements

The typical relationship between I_{CC} and the frequency of operation F is as follows:

$I_{CC_{typ}} = 55 + + 5 * F$ mA where F is in MHz.

When the frequency is reduced below the minimum operating frequency specified in the AC Characteristics table, the internal states of the 80C187 may become indeterminate. The 80C187 clock cannot be stopped; otherwise, I_{CC} would increase significantly beyond what the equation above indicates.

DC Characteristics T_C = 0°C to +85°C, V_{CC} = +5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LOW Voltage	−0.5	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} +0.5	V	
V_{ICL}	Clock Input LOW Voltage	−0.5	+0.8	V	
V_{ICH}	Clock Input HIGH Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output LOW Voltage		0.45	V	
V_{OH}	Output HIGH Voltage	2.4		V	
I_{CC}	Power Supply Current		156 135	mA mA	16 MHz 12.5 MHz
I_{LI}	Input Leakage Current		±10	μA	$0V \le V_{IN} \le V_{CC}$
I_{LO}	I/O Leakage Current		±10	μA	$0.45V \le V_{OUT} \le V_{CC} - 0.45V$
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz
C_O	I/O or Output Capacitance		12	pF	F_C = 1 MHz
C_{CLK}	Clock Capacitance		20	pF	F_C = 1 MHz

AC Characteristics

T_C = 0°C to +85°C, V_{CC} = 5V ±10%
All timings are measured at 1.5V unless otherwise specified

Symbol	Parameter	12.5 MHz		16 MHz		Test Conditions
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	
T_{dvwh} (t6)	Data Setup to $\overline{NPWR}$	43		33		
T_{whdx} (t7)	Data Hold from $\overline{NPWR}$	14		14		
T_{rlrh} (t8)	$\overline{NPRD}$ Active Time	59		54		
T_{wlwh} (t9)	$\overline{NPWR}$ Active Time	59		54		
T_{avwl} (t10)	Command Valid to $\overline{NPWR}$	0		0		
T_{avrl} (t11)	Command Valid to $\overline{NPRD}$	0		0		
T_{mhrl} (t12)	Min Delay from PEREQ Active to $\overline{NPRD}$ Active	40		30		
T_{whax} (t18)	Command Hold from $\overline{NPWR}$	12		8		
T_{rhax} (t19)	Command Hold from $\overline{NPRD}$	12		8		
T_{ivcl} (t20)	$\overline{NPRD}$, $\overline{NPWR}$, RESET to CLK Setup Time	46		38		Note 1
T_{clih} (t21)	$\overline{NPRD}$, $\overline{NPWR}$, RESET from CLK Hold Time	26		18		Note 1
T_{rscl} (t24)	RESET to CLK Setup	21		19		Note 1
T_{clrs} (t25)	RESET from CLK Hold	14		9		Note 1
T_{cmdi} (t26)	Command Inactive Time					
	Write to Write	69		59		
	Read to Read	69		59		
	Read to Write	69		59		
	Write to Read	69		59		

NOTE:
1. This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at a specific CLK edge.

Timing Responses

All timings are measured at 1.5V unless otherwise specified

Symbol	Parameter	12.5 MHz		16 MHz		Test Conditions
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	
T_{rhqz} (t27)	$\overline{NPRD}$ Inactive to Data Float*		18		18	Note 2
T_{rlqv} (t28)	$\overline{NPRD}$ Active to Data Valid		50		45	Note 3
T_{ilbh} (t29)	$\overline{ERROR}$ Active to Busy Inactive	104		104		Note 4
T_{wlbv} (t30)	$\overline{NPWR}$ Active to Busy Active		80		60	Note 4
T_{klml} (t31)	$\overline{NPRD}$ or $\overline{NPWR}$ Active to PEREQ Inactive		80		60	Note 5
T_{rhqh} (t32)	Data Hold from $\overline{NPRD}$ Inactive	2		2		Note 3
T_{rlbh} (t33)	RESET Inactive to BUSY Inactive		80		60	

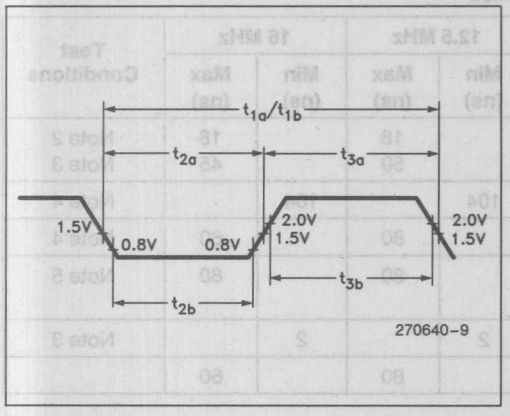
NOTES:
*The data float delay is not tested.
2. The float condition occurs when the measured output current is less than I_{OL} on D_{15}–D_0.
3. D_{15}–D_0 loading: C1 = 100 pF.
4. BUSY loading: C1 = 100 pF.
5. On last data transfer of numeric instruction.

Clock Timings

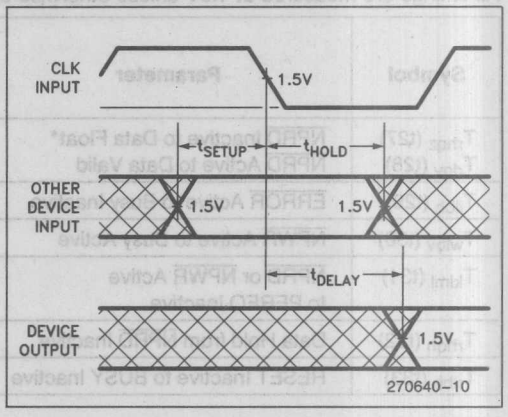
Symbol		Parameter		12.5 MHz		16 MHz*		Test Conditions
				Min (ns)	Max (ns)	Min (ns)	Max (ns)	
T_{clcl}	(t1a)	CLK Period	CKM = 1	80	250	N/A	N/A	Note 6
	(t1B)		CKM = 0	40	125	31.25	125	Note 6
T_{clch}	(t2a)	CLK Low Time	CKM = 1	35		N/A		Note 6
	(t2b)		CKM = 0	9		7		Note 7
T_{chcl}	(t3a)	CLK High Time	CKM = 1	35		N/A		Note 6
	(t3b)		CKM = 0	13		9		Note 8
T_{ch2ch1}	(t4)				10		8	Note 9
T_{ch1ch2}	(t5)				10		8	Note 10

NOTES:
*16 MHz operation is available only in divide-by-2 mode (CKM strapped LOW).
6. At 1.5V
7. At 0.8V
8. At 2.0V
9. CKM = 1: 3.7V to 0.8V at 16 MHz, 3.5V to 1.0V at 12.5 MHz
10. CKM = 1: 0.8V to 3.7V at 16 MHz, 1.0V to 3.5V at 12.5 MHz

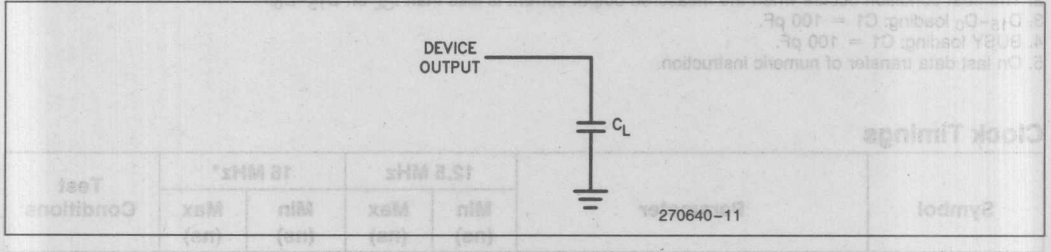
AC DRIVE AND MEASUREMENT POINTS—CLK INPUT



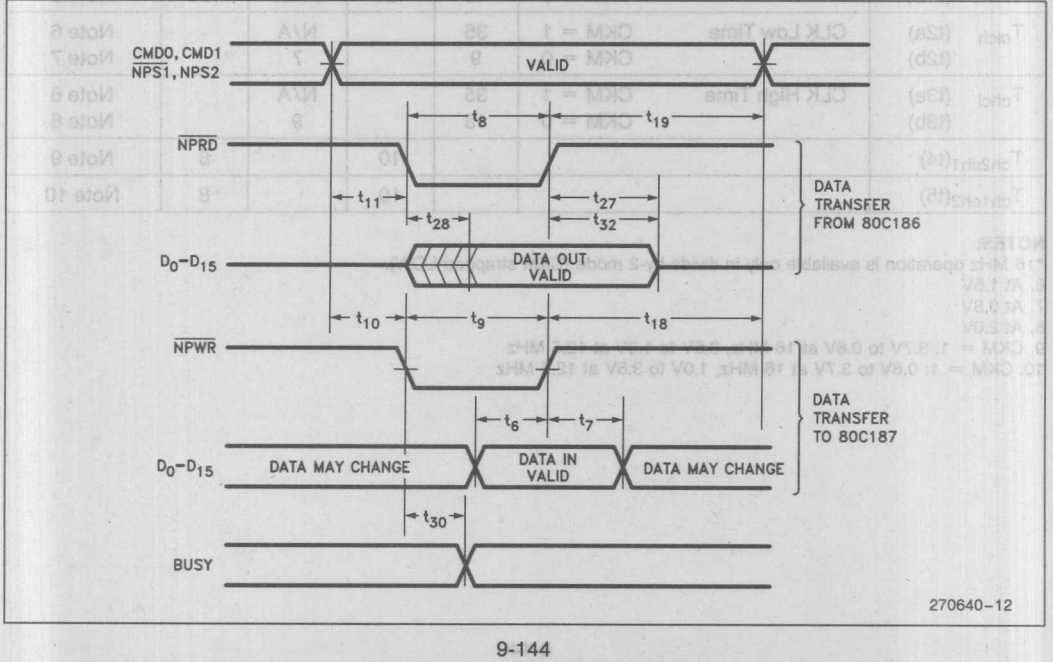
AC SETUP, HOLD, AND DELAY TIME MEASUREMENTS—GENERAL



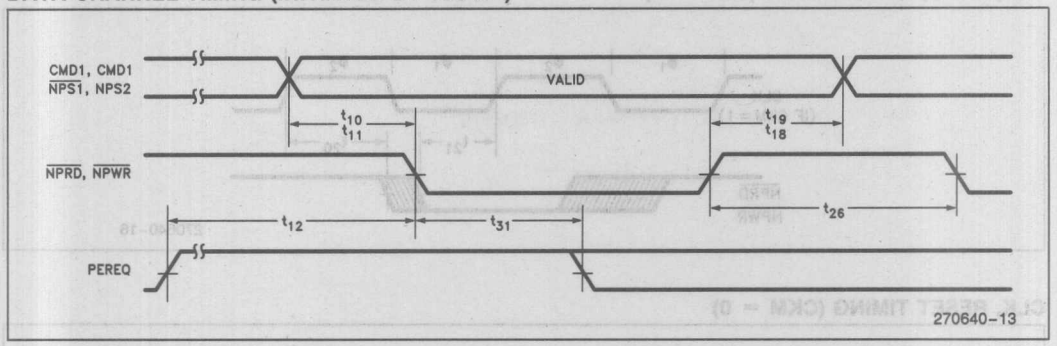
AC TEST LOADING ON OUTPUTS



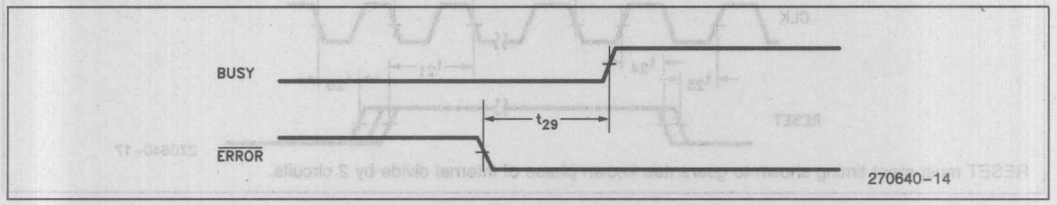
DATA TRANSFER TIMING (INITIATED BY CPU)



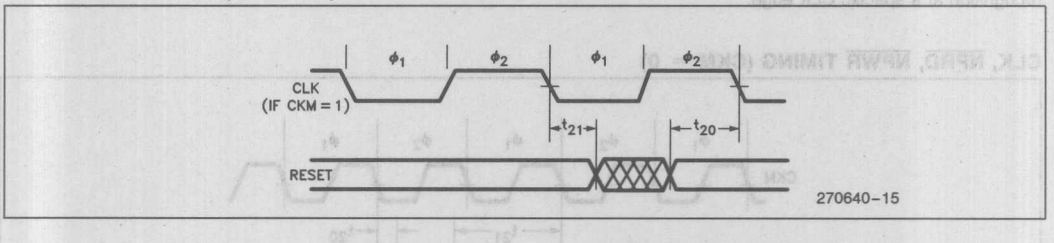
DATA CHANNEL TIMING (INITIATED BY 80C187)



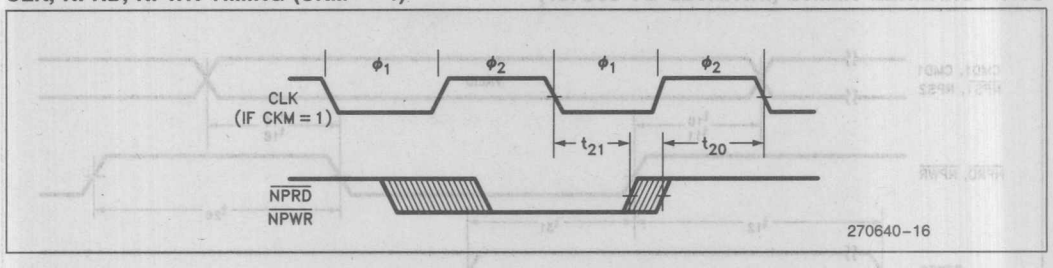
ERROR OUTPUT TIMING



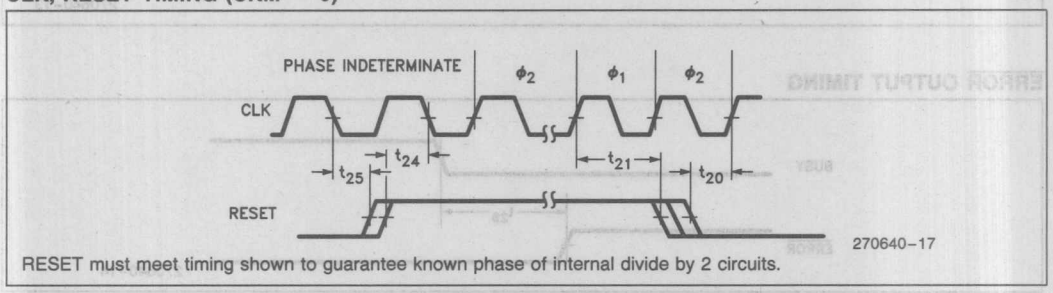
CLK, RESET TIMING (CKM = 1)



CLK, $\overline{NPRD}$, $\overline{NPWR}$ TIMING (CKM = 1)



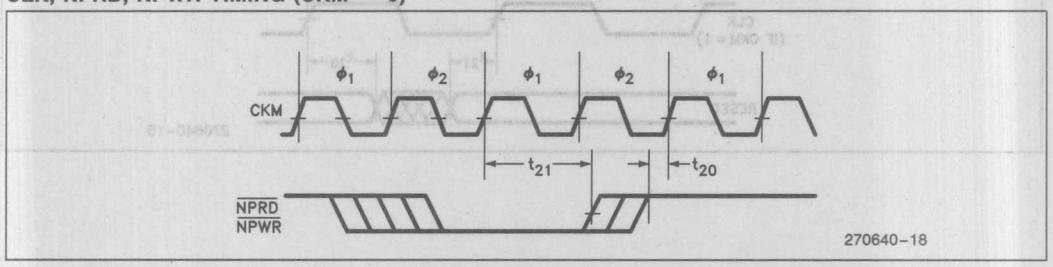
CLK, RESET TIMING (CKM = 0)



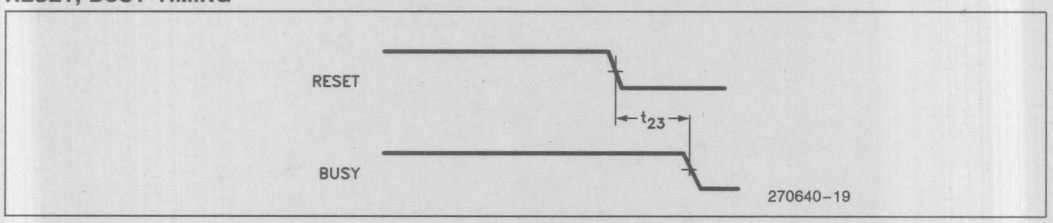
RESET must meet timing shown to guarantee known phase of internal divide by 2 circuits.

NOTE:
RESET, $\overline{NPWR}$, $\overline{NPRD}$ inputs are asynchronous to CLK. Timing requirements are given for testing purposes only, to assure recognition at a specific CLK edge.

CLK, $\overline{NPRD}$, $\overline{NPWR}$ TIMING (CKM = 0)



RESET, BUSY TIMING



80C187 EXTENSIONS TO THE CPU's INSTRUCTION SET

Instructions for the 80C187 assume one of the five forms shown in Table 11. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the CPU's addressing modes.

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of CPU instructions (refer to Programmer's Reference Manual for the CPU). The DISP (displacement) is optionally present in instructions that have MOD and R/M fields. Its presence depends on the values of MOD and R/M, as for instructions of the CPU.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD requests delaying processor access to the bus; and that no exceptions are detected during instruction execution. Timings are given in internal 80C187 clocks and include the time for opcode and data transfer between the CPU and the NPX. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

Table 11. Instruction Formats

	Instruction									Optional Field
	First Byte				Second Byte					
1	11011	OPA		1	MOD		1	OPB	R/M	DISP
2	11011	MF		OPA	MOD		OPB *		R/M	DISP
3	11011	d	P	OPA	1	1	OPB *		ST (i)	
4	11011	0	0	1	1	1	1	OP		
5	11011	0	1	1	1	1	1	OP		
	15–11	10	9	8	7	6	5	4 3	2 1 0	

NOTES:

OP = Instruction opcode, possibly split into two fields OPA and OPB

MF = Memory Format
- 00— 32-Bit Real
- 01— 32-Bit Integer
- 10— 64-Bit Real
- 11— 16-Bit Integer

d = Destination
- 0— Destination is ST(0)
- 0— Destination is ST(i)

R XOR d = 0— Destination (op) Source

R XOR d = 1— Source (op) Destination

*In FSUB and FDIV, the low-order bit of OPB is the R (reversed) bit

P = Pop
- 0— Do not pop stack
- 1— Pop stack after operation

ESC = 11011

ST(i) = Register Stack Element *i*
- 000 = Stack Top
- 001 = Second Stack Element
- •
- •
- •
- 111 = Eighth Stack Element

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–3	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load[a]							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	DISP	40	65–72	59	67–71
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	DISP		90–101		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	DISP		74		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	DISP		296–305		
ST(i) to ST(0)	ESC 001	11000 ST(i)			16		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	DISP	58	93–107	73	80–93
ST(0) to ST(i)	ESC 101	11010 ST(i)			13		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	DISP	58	93–107	73	80–93
ST(0) to long integer memory	ESC 111	MOD 111 R/M	DISP		116–133		
ST(0) to extended real	ESC 011	MOD 111 R/M	DISP		83		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	DISP		542–564		
ST(0) to ST(i)	ESC 101	11001 ST (i)			14		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			20		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	DISP	48	78–85	67	77–81
ST(i) to ST(0)	ESC 000	11010 ST(i)			26		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	DISP	48	78–85	67	77–81
ST(i) to ST(0)	ESC 000	11011 ST(i)			28		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			28		
FTST = Test ST(0)	ESC 001	1110 0100			30		
FUCOM = Unordered compare	ESC 101	11100 ST(i)			26		
FUCOMP = Unordered compare and pop	ESC 101	11101 ST(i)			28		
FUCOMPP = Unordered compare and pop twice	ESC 010	1110 1001			28		
FXAM = Examine ST(0)	ESC 001	11100101			32-40		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)	ESC 001	1110 1110			22		
FLD1 = Load +1.0 into ST(0)	ESC 001	1110 1000			26		
FLDPI = Load pi into ST(0)	ESC 001	1110 1011			42		
FLDL2T = Load $\log_2(10)$ into ST(0)	ESC 001	1110 1001			42		

Shaded areas indicate instructions not available in 8087.

NOTE:
a. When loading single- or double-precision zero from memory, add 5 clocks.

80C187 Extensions to the 80C186 Instruction Set (Continued)

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–3	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS (Continued)							
FLDL2E = Load $\log_2(e)$ into ST(0)	ESC 001	1110 1010			42		
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100			43		
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101			43		
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	DISP	44–52	77–92	65–73	77–91
ST(i) and ST(0)	ESC d P 0	11000 ST(i)			25–33[b]		
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	DISP	44–52	77–92	65–73	77–91[c]
ST(i) and ST(0)	ESC d P 0	1110 R R/M			28–36[d]		
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	DISP	47–57	81–102	68–93	82–93
ST(i) and ST(0)	ESC d P 0	1100 1 R/M			31–59[e]		
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	DISP	108	140–147[f]	128	142–146[g]
ST(i) and ST(0)	ESC d P 0	1111 R R/M			90[h]		
FSQRT[i] = Square root	ESC 001	1111 1010			124–131		
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101			69–88		
FPREM = Partial remainder of ST(0) ÷ ST(1)	ESC 001	1111 1000			76–157		
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101			97–187		
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100			68–82		
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100			72–78		
FABS = Absolute value of ST(0)	ESC 001	1110 0001			24		
FCHS = Change sign of ST(0)	ESC 001	1110 0000			26–27		

Shaded areas indicate instructions not available in 8087.

NOTES:
b. Add 3 clocks to the range when d = 1.
c. Add 1 clock to **each** range when R = 1.
d. Add 3 clocks to the range when d = 0.
e. typical = 54 (When d = 0, 48–56, typical = 51).
f. Add 1 clock to the range when R = 1.
g. 153–159 when R = 1.
h. Add 3 clocks to the range when d = 1.
i. $-0 \leq ST(0) \leq +\infty$.

80C187 Extensions to the 80C186 Instruction Set (Continued)

Instruction	Encoding			Clock Count Range
	Byte 0	Byte 1	Optional Bytes 2–3	
TRANSCENDENTAL				
FCOS = Cosine of ST(0)	ESC 001	1111 1111		125–774[j]
FPTAN[k] = Partial tangent of ST(0)	ESC 001	1111 0010		193–499[j]
FPATAN = Partial arctangent	ESC 001	1111 0011		316–489
FSIN = Sine of ST(0)	ESC 001	1111 1110		124–773[j]
FSINCOS = Sine and cosine of ST(0)	ESC 001	1111 1011		196–811[j]
F2XM1[l] = $2^{ST(0)} - 1$	ESC 001	1111 0000		213–478
FYL2X[m] = $ST(1) * \log_2(ST(0))$	ESC 001	1111 0001		122–540
FYL2XP1[n] = $ST(1) * \log_2(ST(0) + 1.0)$	ESC 001	1111 1001		259–549
PROCESSOR CONTROL				
FINIT = Initialize NPX	ESC 011	1110 0011		35
FSTSW AX = Store status word	ESC 111	1110 0000		17
FLDCW = Load control word	ESC 001	MOD 101 R/M	DISP	23
FSTCW = Store control word	ESC 001	MOD 111 R/M	DISP	21
FSTSW = Store status word	ESC 101	MOD 111 R/M	DISP	21
FCLEX = Clear exceptions	ESC 011	1110 0010		13
FSTENV = Store environment	ESC 001	MOD 110 R/M	DISP	146
FLDENV = Load environment	ESC 001	MOD 100 R/M	DISP	113
FSAVE = Save state	ESC 101	MOD 110 R/M	DISP	550
FRSTOR = Restore state	ESC 101	MOD 100 R/M	DISP	482
FINCSTP = Increment stack pointer	ESC 001	1111 0111		23
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		24
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		20
FNOP = No operations	ESC 001	1101 0000		14

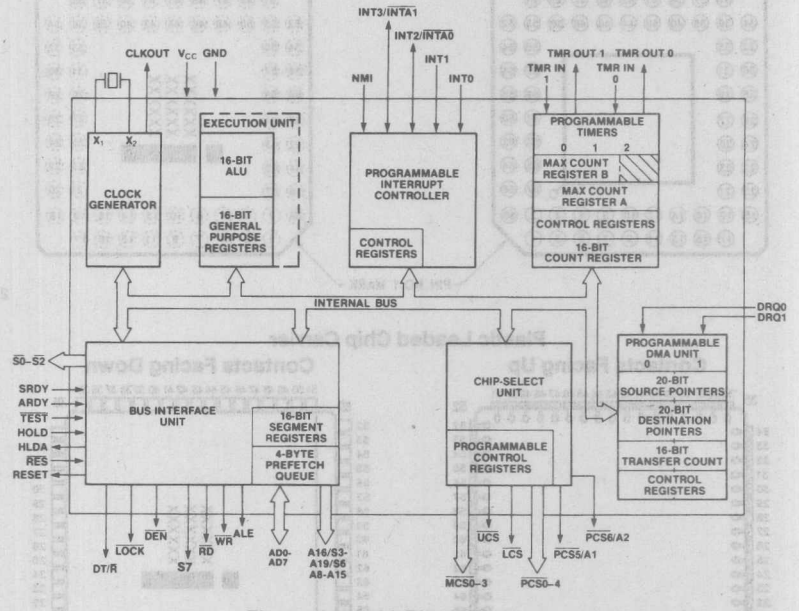
Shaded areas indicate instructions not available in 8087.

NOTES:
j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 78 clocks may be needed to reduce the operand.
k. $0 \leq |ST(0)| < 2^{63}$.
l. $-1.0 \leq ST(0) \leq 1.0$.
m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.
n. $0 \leq |ST(0)| < (2 - \sqrt{(2)})/2$, $-\infty < ST(1) < +\infty$.

80188
HIGH INTEGRATION 8-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - — Enhanced 8086-2 CPU
  - — Clock Generator
  - — 2 Independent DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-Bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- **16-Bit Internal Architecture with 8-Bit Data Bus Interface**
- **High-Performance 8 MHz Processor**
  - — At 8 MHz Provides 2 Times the Performance of the Standard 8088
  - — 2 MByte/Sec Bus Bandwidth Interface @8 MHz
- **Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086/8088 Software**
  - — 10 New Instruction Types
- **Complete System Development Support**
  - — Development Software: ASM86 Assembler, PL/M-86, Pascal-86, Fortran-86, C-86, and System Utilities
  - — In-Circuit-Emulator (I²ICE™-186/188)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**
- **Available in 68 Pin:**
  - — Ceramic Leadless Chip Carrier (LCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Plastic Leaded Chip Carrier (PLCC)

  (See Packaging Outlines and Dimensions, Order #231369)
- **Available in EXPRESS**
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)



210706-1

Figure 1. 80188 Block Diagram

October 1988
Order Number: 210706-010

The Intel 80188 is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The 80188 effectively combines 15–20 of the most common 8088 system components onto one. The 80188 provides two times greater throughput than the standard 5 MHz 8088. The 80188 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

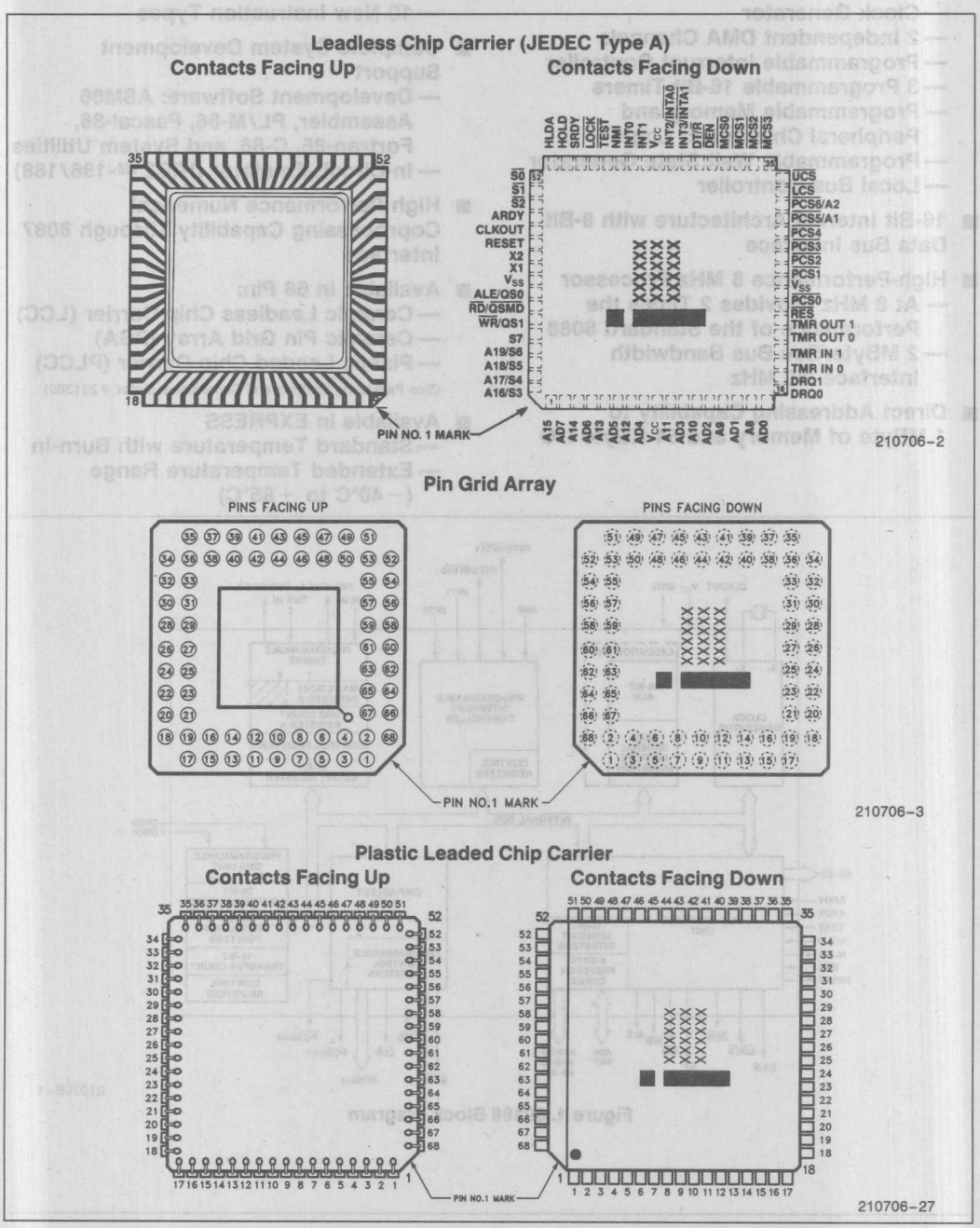


Figure 2. 80188 Pinout Diagram

Table 1. 80188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	9 43	I I	**SYSTEM POWER:** +5 volt power supply.
V_{SS}	26 60	I I	**SYSTEM GROUND**
RESET	57	O	**RESET OUTPUT:** Indicates that the 80188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1 X2	59 58	I O	**CRYSTAL INPUTS:** X1 and X2 provide external connections for a fundamental mode parallel resonant crystal for the internal oscillator. Instead of using a crystal, an external clock may be applied to X1 while minimizing stray capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	**CLOCK OUTPUT:** Provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT.
$\overline{RES}$	24	I	**PROCESSOR RESET:** Causes the 80188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80188 clock. The 80188 begins fetching instructions approximately 6½ clock cycles after $\overline{RES}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{RES}$ held low. $\overline{RES}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then float them.
$\overline{TEST}$	47	I	**$\overline{TEST}$:** Is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80188 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0 TMR IN 1	20 21	I I	**TIMER INPUTS:** Are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0 TMR OUT 1	22 23	O O	**TIMER OUTPUTS:** Are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	**DMA REQUEST:** Is asserted HIGH by an external device when it is ready for DMA Channel 0 or 1 to perform a transfer. These signals are level-triggered and internally synchronized.
NMI	46	I	**NON-MASKABLE INTERRUPT:** Causes a Type 2 interrupt. An NMI transition from LOW to HIGH is latched and synchronized internally, and initiates the interrupt at the next instruction boundary. NMI must be asserted for at least one clock. The Non-Maskable Interrupt cannot be avoided by programming.
INT0 INT1 INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45 44 42 41	I I I/O I/O	**MASKABLE INTERRUPT REQUESTS:** Can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
A19/S6 A18/S5 A17/S4 A16/S3	65 66 67 68	O O O O	**ADDRESS BUS OUTPUTS (16–19) and BUS CYCLE STATUS (3–6):** Indicate the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:
			S6: Low = Processor Cycle; High = DMA Cycle
			S3, S4, and S5 are defined as LOW during T_2–T_4. The status pins float during HOLD/HLDA.
AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0	2 4 6 8 11 13 15 17	I/O I/O I/O I/O I/O I/O I/O I/O	**ADDRESS/DATA BUS (0-7):** Signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH.
A15 A14 A13 A12 A11 A10 A9 A8	1 3 5 7 10 12 14 16	O O O O O O O O	**ADDRESS-ONLY BUS (8-15):** Containing valid address from T_1-T_4. The bus is active HIGH.
S7	64	O	This signal is HIGH to indicate that the 80188 has an 8-bit data bus. S7 floats during HOLD.
ALE/QS0	61	O	**ADDRESS LATCH ENABLE/QUEUE STATUS 0:** Is provided by the 80188 to latch the address. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the 8088. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8088. Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	**WRITE STROBE/QUEUE STATUS 1:** Indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during HOLD. When the 80188 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.
			QS1 / QS0 / Queue Operation: 0 / 0 / No Queue Operation 0 / 1 / First Opcode Byte Fetched from the Queue 1 / 1 / Subsequent Byte Fetched from the Queue 1 / 0 / Empty the Queue

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$/QSMD	62	I/O	**READ STROBE:** Is an active LOW signal which indicates that the 80188 is performing a memory or I/O read cycle. It is guaranteed not to go LOW before the A/D bus is floated. An internal pull-up ensures that $\overline{RD}$ is HIGH during RESET. Following RESET the pin is sampled to determine whether the 80188 is to provide ALE, $\overline{RD}$, and $\overline{WR}$, or queue status information. To enable Queue Status Mode, $\overline{RD}$ must be connected to GND. $\overline{RD}$ will float during bus hold.
ARDY	55	I	**ASYNCHRONOUS READY:** Informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT and is active HIGH. The falling edge of ARDY must be synchronized to the 80188 clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.
SRDY	49	I	**SYNCHRONOUS READY:** Informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows a relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchronize the ARDY input signal. Connecting SRDY high will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the ARDY pin.
$\overline{LOCK}$	48	O	**$\overline{LOCK}$:** Output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of that instruction. No instruction prefetching will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	52 53 54	O O O	**BUS CYCLE STATUS $\overline{S0}$–$\overline{S2}$:** Are encoded to provide bus-transaction information:

80188 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	**HOLD:** Indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80188 clock. The 80188 will issue a HLDA in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80188 will lower HLDA. When the 80188 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	**UPPER MEMORY CHIP SELECT:** Is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	**LOWER MEMORY CHIP SELECT:** Is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$ $\overline{MCS1}$ $\overline{MCS2}$ $\overline{MCS3}$	38 37 36 35	O O O O	**MID-RANGE MEMORY CHIP SELECT SIGNALS:** Are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$ $\overline{PCS2}$ $\overline{PCS3}$ $\overline{PCS4}$	25 27 28 29 30	O O O O O	**PERIPHERAL CHIP SELECT SIGNALS 0–4:** Are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	**PERIPHERAL CHIP SELECT 5 or LATCHED A1:** May be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	**PERIPHERAL CHIP SELECT 6 or LATCHED A2:** May be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	**DATA TRANSMIT/RECEIVE:** Controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the 80188. When HIGH the 80188 places write data on the data bus.
$\overline{DEN}$	39	O	**DATA ENABLE:** Is provided as a data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80188. The 80188 is a very high integration 8-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8088. The 80188 is object code compatible with the 8086, 8088 microprocessors and adds 10 new instruction types to the 8086, 8088 instruction set.

80188 BASE ARCHITECTURE

The 8086, 8088, 80186, 80188 and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set

The 80188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

GENERAL REGISTERS

Eight 16-bit general purpose registers may be used for arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

SEGMENT REGISTERS

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

BASE AND INDEX REGISTERS

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

STATUS AND CONTROL REGISTERS

Two 16-bit special purpose registers record or alter certain aspects of the 80188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

STATUS WORD DESCRIPTION

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80188 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

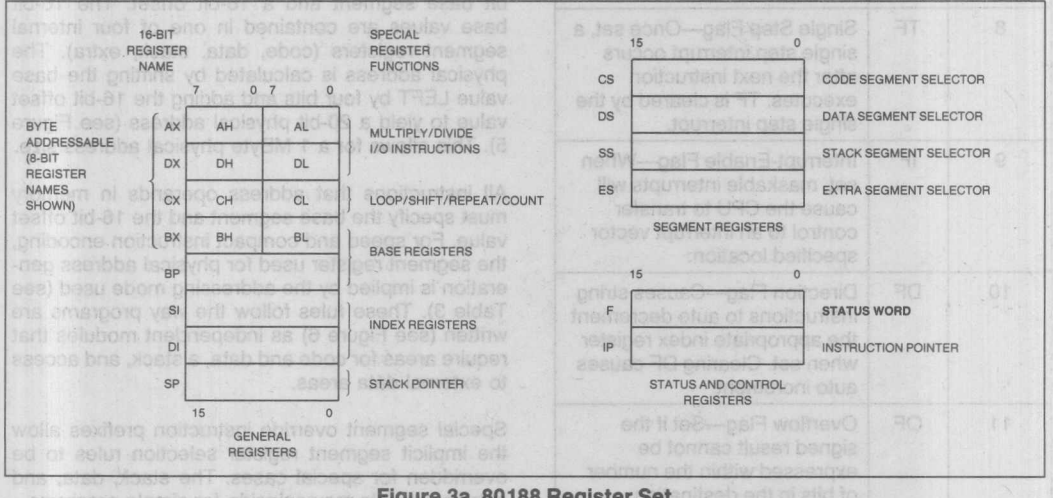


Figure 3a. 80188 Register Set

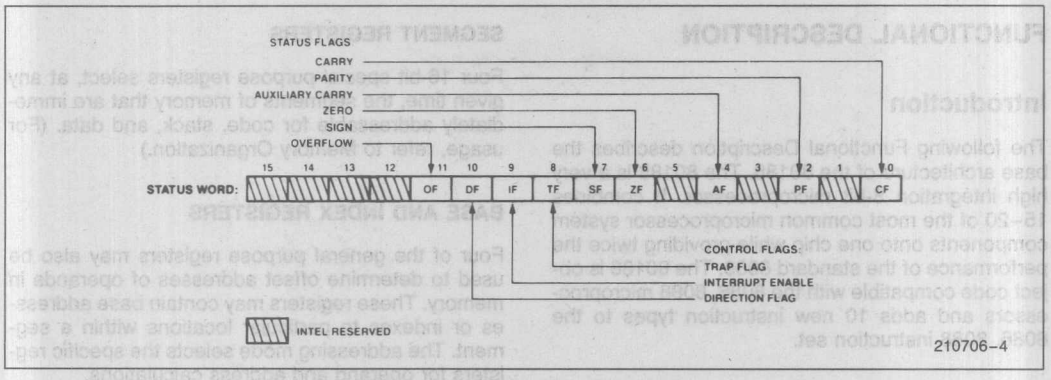


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

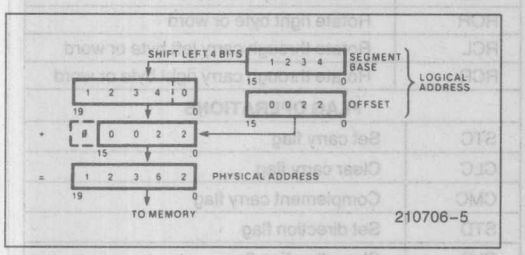


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

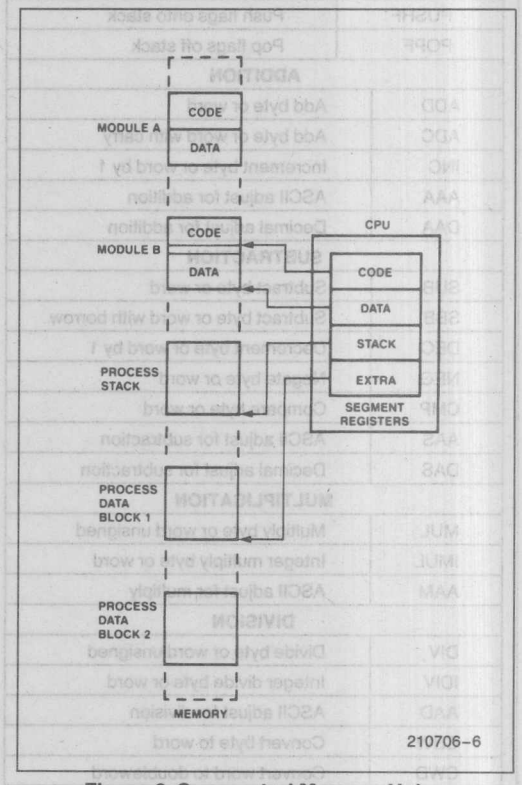


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using an 8087 Numeric Data Coprocessor with the 80188.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using an 8087 Numeric Data Coprocessor with the 80188.)

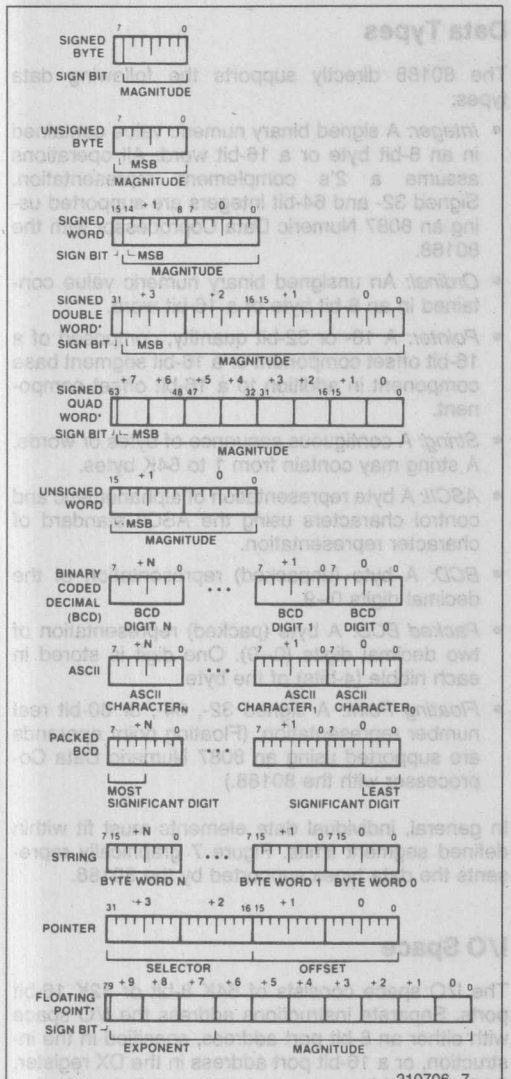
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported using an 8078 Numeric Data Coprocessor with the 80188.

Figure 7. 80188 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80188 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80188 Interrupt Vectors

Interrupt Name	Vector Type	Vector Address	Default Priority	Related Instructions	Applicable Notes
Divide Error Exception	0	00H	1	DIV, IDIV	1
Single Step Interrupt	1	04H	1A	All	2
Non-Maskable Interrupt (NMI)	2	08H	1	All	
Breakpoint Interrupt	3	0CH	1	INT	1
INT0 Detected Overflow Exception	4	10H	1	INTO	1
Array Bounds Exception	5	14H	1	BOUND	1
Unused Opcode Exception	6	18H	1	Undefined Opcodes	1
ESC Opcode Exception	7	1CH	1	ESC Opcodes	1, 3
Timer 0 Interrupt	8	20H	2A		4
Timer 1 Interrupt	18	48H	2B		4
Timer 2 Interrupt	19	4CH	2C		4
Reserved	9	24H	3		
DMA 0 Interrupt	10	28H	4		
DMA 1 Interrupt	11	2CH	5		
INT0 Interrupt	12	30H	6		
INT1 Interrupt	13	34H	7		
INT2 Interrupt	14	38H	8		
INT3 Interrupt	15	3CH	9		
Reserved	16, 17	40H, 44H			
Reserved	20–31	50H . . . 7CH			

NOTES:
Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
1. Generated as a result of an instruction execution.
2. Performed in same manner as 8088.
3. An ESC opcode will cause a trap if the power bit is set in the peripheral control block relocation register.
4. All three timers constitute one source of request to the interrupt controller. As such, they share the same priority level with respect to other interrupt sources. However, the timers have a defined priority order among themselves (2A > 2B > 2C).

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE 4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80188 provides maskable hardware in-

terrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts are shown in Table 4. Software enables these inputs by setting the Interrupt Flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80188 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80188 COMPARED TO THE 80186

The 80188 CPU is an 8-bit processor designed around the 80186 internal structure. Most internal functions of the 80188 are identical to the equivalent 80186 functions. The 80188 handles the external bus the same way the 80186 does with the distinction of handling only 8 bits at a time. Sixteen bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80188 and the 80186 are outlined below. Internally, there are three differences between the 80188 and the 80186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80188, whereas the 80186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80186 waits until a 2-byte space is available.
- The internal execution time of the instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the speed of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80188 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80188 and 80186 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally well on an 80188 or an 80186.

The hardware interface of the 80188 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes.

- A8–A15—These pins are only address outputs on the 80188. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 80188 and has been eliminated.

80188 Clock Generator

The 80188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80188. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80188. The recommended crystal configuration is shown in Figure 8.

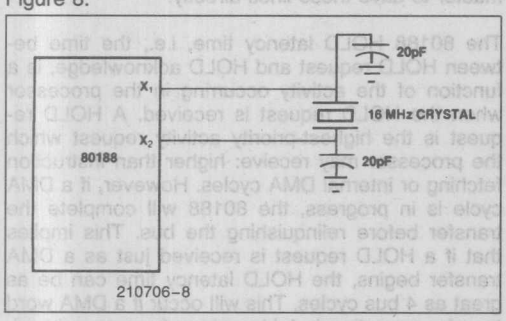


Figure 8. Recommended 80188 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_L (Load Capacitance):	20 pf ±2 pf
Drive Level:	1 mW max

Clock Generator

The 80188 clock generator provides the 50% duty cycle processor clock for the 80188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3, and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used for full synchronization of a rising ARDY signal. A high-to-low transition on ARDY may be used as an indication of the not ready condition but it must be performed synchronously to CLKOUT **either** in the middle of T_2, T_3, **or** T_W, or at the falling edge of T_3 or T_W.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3, and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80188 provides both a $\overline{RES}$ input pin and a synchronized RESET output pin for use with other system components. The $\overline{RES}$ input pin on the 80188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to approximately two and one-half clocks behind $\overline{RES}$.

Multiple 80188 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets

both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides outputs that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80188 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory or I/O to the 80188 or to strobe data from the 80188 to memory or I/O. The ALE line provides a strobe to latch the address when it is valid. The 80188 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80188 generates two control signals for external transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. External circuitry must arbitrate which external device will gain control of the bus when there is more than one alternate local bus master. When the 80188 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD7, A8–A19, $\overline{S7}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80188 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

During RESET the local bus controller will perform the following actions:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the inactive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Three-state AD0–7, A8–19, S7, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80188 integrated peripherals are controlled by 16-bit registers contained within an internal 256-byte control block. The control block may be mapped into either memory or I/O space. Internal logic will recognize control block addresses and respond to bus cycles. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{7-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80188 CPU at any time

The control block base address is programmed by a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space. If the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH which maps the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

CHIP-SELECT/READY GENERATION LOGIC

The 80188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80188 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

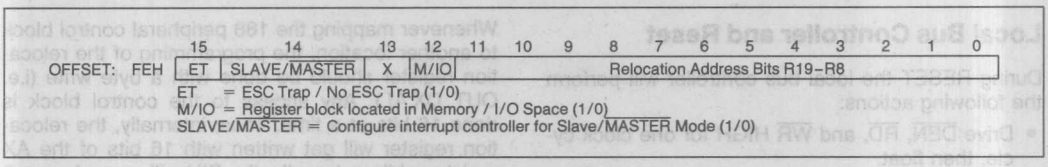


Figure 9. Relocation Register

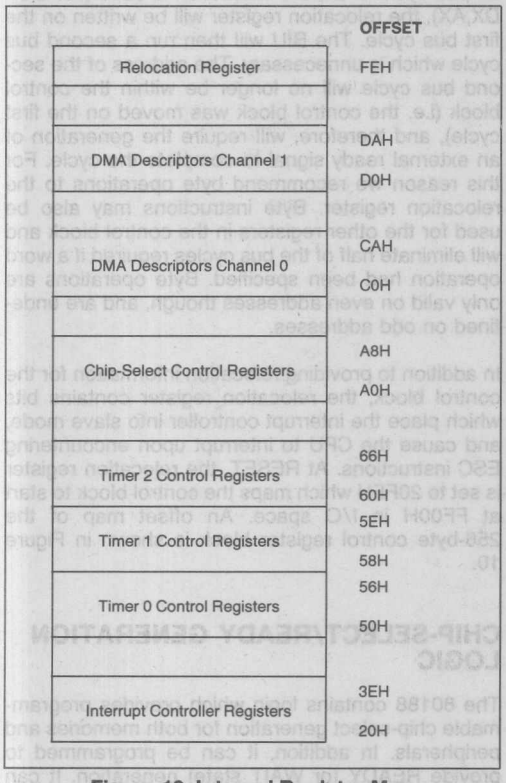


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

Upper Memory $\overline{CS}$

The 80188 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0 = R1 = R2 = 0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

The internal generation of any 20-bit address whose upper 16 bits are equal to or greater than the UMCS value (with bits 0–5 as "0") asserts $\overline{UCS}$. UMCS bits R2–R0 specify the ready mode for the area of memory defined by the chip select register, as explained later.

Lower Memory $\overline{CS}$

The 80188 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12) at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will assert $\overline{LCS}$. LMCS register bits R2–R0 specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80188 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located within the 80188 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. If the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

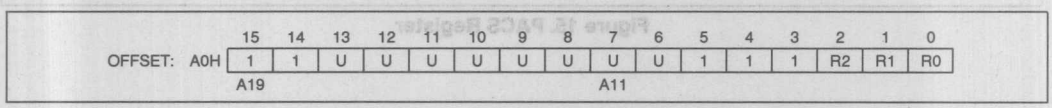


Figure 11. UMCS Register

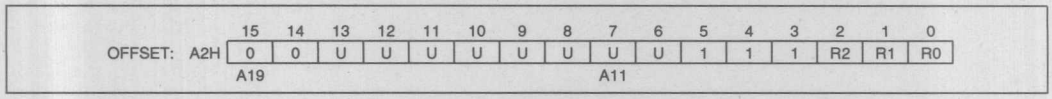


Figure 12. LMCS Register

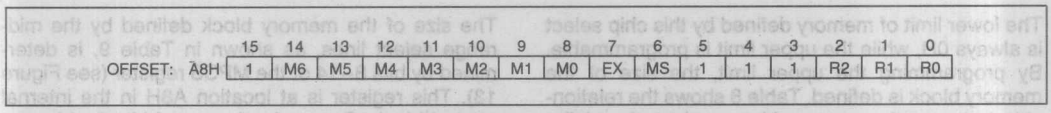


Figure 13. MPCS Register

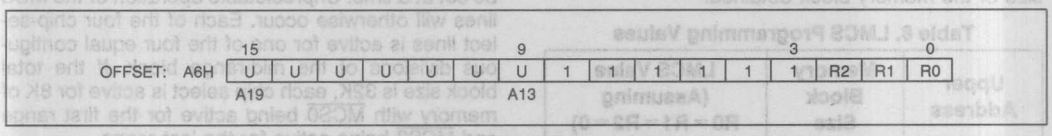


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all four mid-range chip selects.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The 80188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. The base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1 and A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0 and A1 pins used for selecting internal registers of 8-bit peripheral chips.

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). The register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

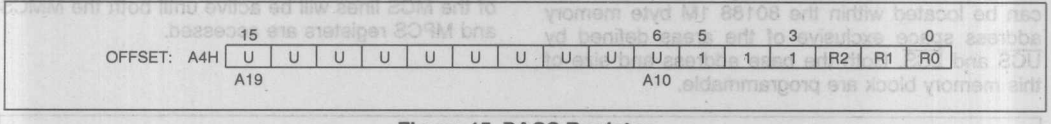


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

$\overline{PCS}$ Line	Active between Locations
$\overline{PCS0}$	PBA —PBA + 127
$\overline{PCS1}$	PBA + 128—PBA + 255
$\overline{PCS2}$	PBA + 256—PBA + 383
$\overline{PCS3}$	PBA + 384—PBA + 511
$\overline{PCS4}$	PBA + 512—PBA + 639
$\overline{PCS5}$	PBA + 640—PBA + 767
$\overline{PCS6}$	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). The register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 specify the READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80188 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). For example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon RESET, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to insert 3 wait states in conjunction with external READY (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA Channels

The 80188 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a data transfer rate of one MByte/sec at 8 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Count Register, and a 16-bit Control Word.

The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

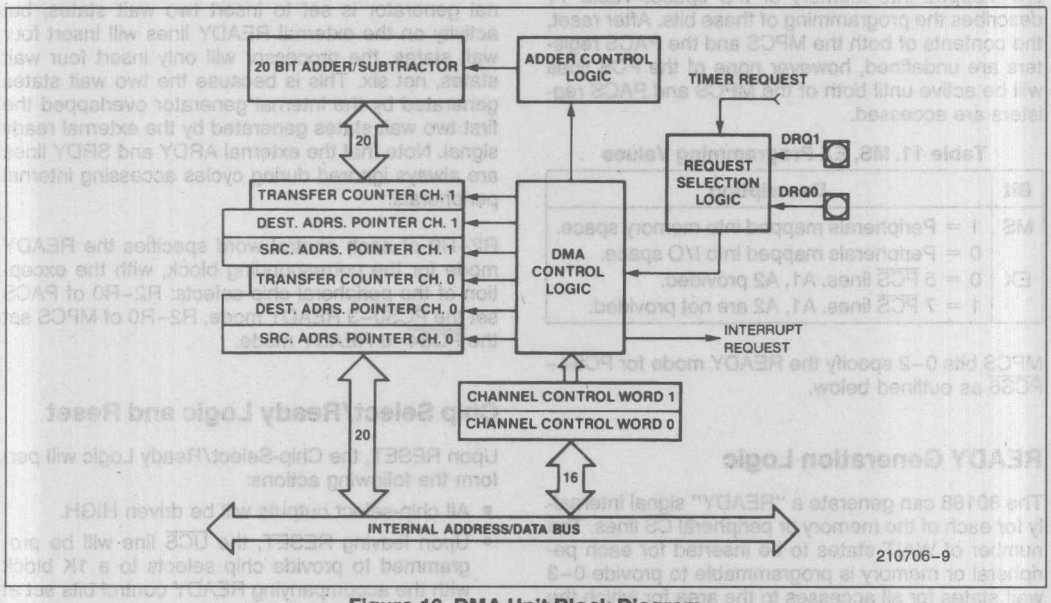


Figure 16. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/$\overline{IO}$	DESTINATION DEC	DESTINATION INC	M/$\overline{IO}$	SOURCE DEC	SOURCE INC	TC	INT	SYN	P	TDRQ	X		CHG/$\overline{NOCHG}$	ST/$\overline{STOP}$	X

X = Don't Care

Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

DEST: M/$\overline{IO}$ Destination pointer is in memory (1) or I/O (0) space.

DEC Decrement destination pointer by 1 after each transfer.

INC Increment destination pointer by 1 after each transfer.

If both INC and DEC are specified, the pointer will not be changed after each cycle.

SOURCE: M/$\overline{IO}$ Source pointer is in memory (1) or I/O (0) space.

DEC Decrement source pointer by 1 after each transfer.

INC Increment source pointer by 1 after each transfer.

If both INC and DEC are specified, the pointer will not be changed after each cycle.

TC: If set, DMA will terminate when the contents of the transfer count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point. If cleared, the DMA controller will decrement the transfer count register for each DMA cycle, but DMA transfers will not stop when the transfer count register reaches zero.

INT: Enable interrupts to CPU upon transfer count termination.

SYN: 00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source Synchronization.

10 Destination Synchronization.

11 Unused.

P: Channel priority relative to other channel.

0 Low priority.

1 High priority.

Channels will alternate cycles if both are set at the same priority level.

TDRQ: Enable/Disable (1/0) DMA requests from Timer 2.

CHG/$\overline{NOCHG}$: Change/Do Not Change (1/0) the ST/$\overline{STOP}$ bit. If this bit is set when writing the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be read as 0.

ST/$\overline{STOP}$: Start/Stop (1/0) Channel.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). The register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). When source synchronized or unsynchronized transfers are performed, data will not be fetched from the source address until the destination device signals that it is ready to receive it. Also, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another destination synchronized DMA cycle will begin after two processor clocks. This allows the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. Table 14 shows the maximum DMA transfer rates.

Table 14. Maximum DMA Transfer Rates @ 8 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	1.0 MBytes/sec	1.0 MBytes/sec
Source Synch	1.0 MBytes/sec	1.0 MBytes/sec
Destination Synch	0.67 MBytes/sec	0.80 MBytes/sec

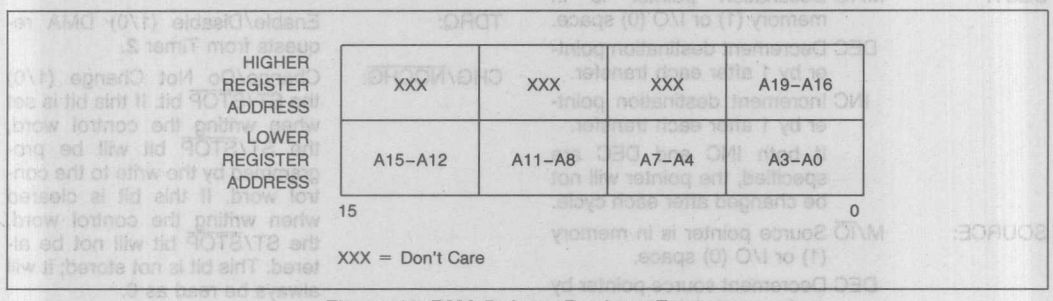


Figure 18. DMA Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed to give one channel priority over the other, or they may be programmed to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; also, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/$\overline{\text{STOP}}$ bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also be generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before the ST/$\overline{\text{STOP}}$ bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/$\overline{\text{NOCHG}}$ bit is cleared when the control register is written, the ST/$\overline{\text{STOP}}$ bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The ST/$\overline{\text{STOP}}$ bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, the third timer can be used as a prescaler to the other two, or as a DMA request source.

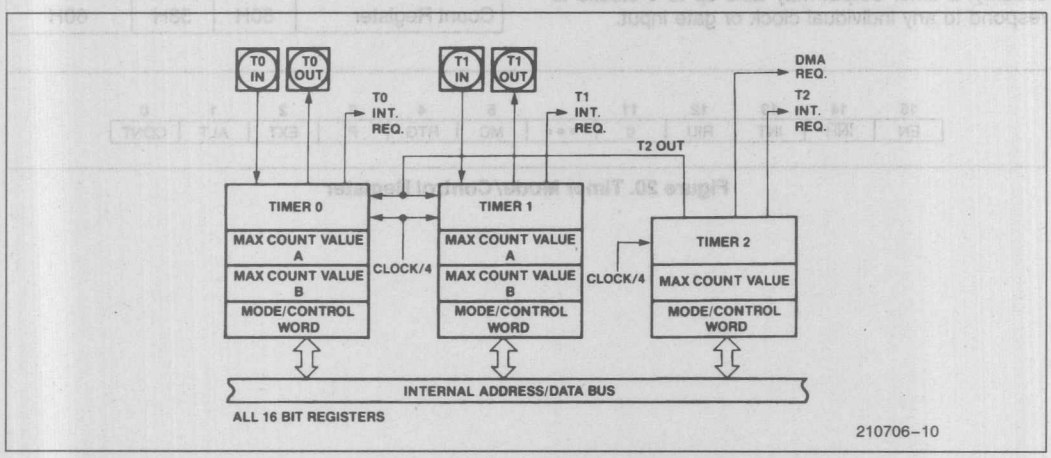


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

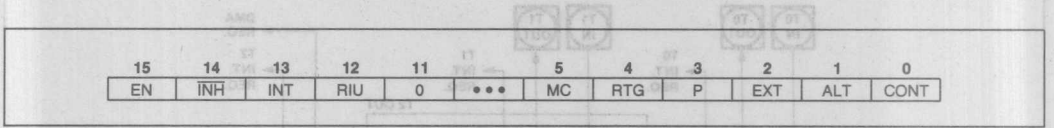


Figure 20. Timer Mode/Control Register

EN

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{\text{INH}}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{\text{INH}}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

RIU

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

MC

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RTG

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

P

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

EXT

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

ALT

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of

the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The contents of this register may be read or written by the processor at any time. If the register is written while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. A timer resets when the timer count register equals the max count value being used. If the timer count register or the max count register is changed so that the max count is less than the timer count, the timer does not immediately reset. Instead, the timer counts up to 0FFFFH, "wraps around" to zero, counts up to the max count value, and then resets.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- For Timers 0 and 1, the RIU bits are reset to zero and the ALT bits are set to one. This results in the Timer Out pins going HIGH.

INTERRUPT CONTROLLER

The 80188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80188 interrupt controller has its own control register that sets the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80188 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

Five pins are provided for external interrupt sources. One of these pins is NMI, the non-maskable interrupt. NMI is generally used for unusual events such as power-fail interrupts. The other four pins may be configured in any of the following ways:

- As four interrupt input lines with internally generated interrupt vectors.
- As an interrupt line and interrupt acknowledge line pair (Cascade Mode) with externally generated interrupt vectors plus two interrupt input lines with internally generated vectors.
- As two pairs of interrupt/interrupt acknowledge lines (Cascade Mode) with externally generated interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80188 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

FULLY NESTED MODE

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled, yet be suspended only by interrupts of higher priority than the in-service interrupt.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is executed at the end of the service routine just before the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

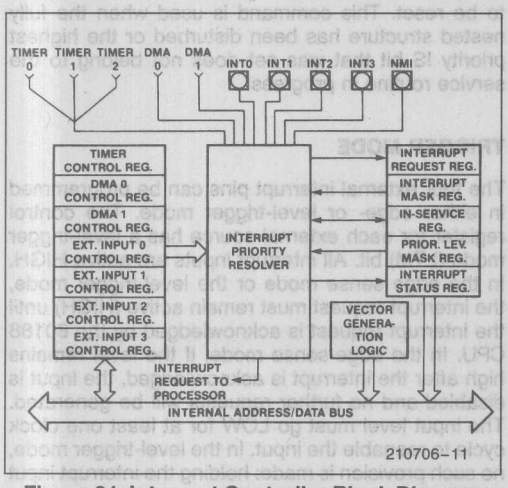


Figure 21. Interrupt Controller Block Diagram

CASCADE MODE

The 80188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80188 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

SPECIAL FULLY NESTED MODE

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80188 controller until the 80188 in-service bit is reset. In special fully nested mode, the 80188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be

set, however, to inhibit interrupts from other lower-priority 80188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the IS register in the external master 8259A is required to determine if there is more than one bit set. If so, the IS bit in the 80188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

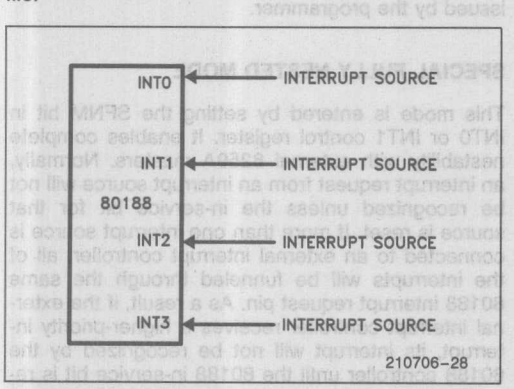


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Master Mode Features

PROGRAMMABLE PRIORITY

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, other interrupt requests can be serviced.

END-OF-INTERRUPT COMMAND

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

TRIGGER MODE

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

INTERRUPT VECTORING

The 80188 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

IN-SERVICE REGISTER

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command.

INTERRUPT REQUEST REGISTER

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits are set when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

MASK REGISTER

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corresponding to a particular source masks the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

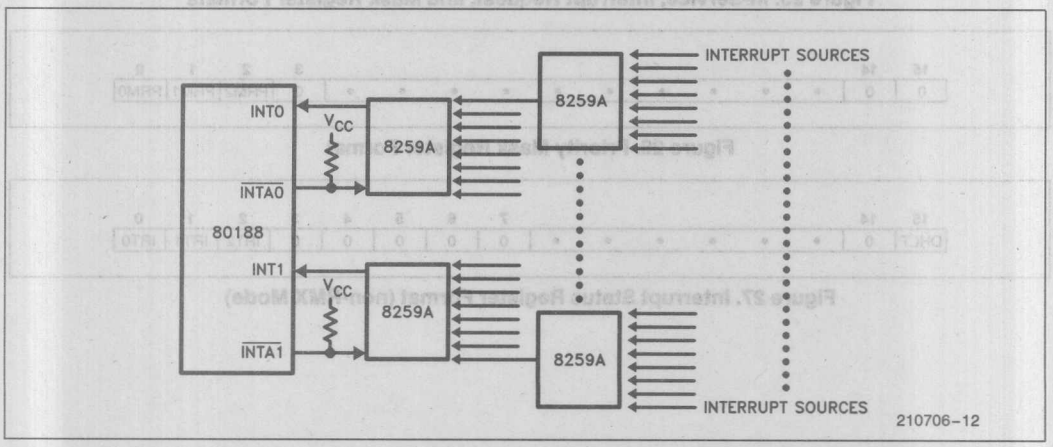


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

PRIORITY MASK REGISTER

This register masks all interrupts below a particular interrupt priority level. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

INTERRUPT STATUS REGISTER

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. This bit allows prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits differentiate between timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14	•	•	•	10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14	•	•	•	•	•	•	•	•	•	•	3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	•	0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14	•	•	•	•	•	•	7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (non-RMX Mode)

TIMER, DMA 0, 1; CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 CONTROL REGISTERS

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest priority = 000, lowest priority = 111.

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct.

SFNM: Special fully nested mode bit, 1 = SFNM.

EOI REGISTER

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80188 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:
To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

15	14	•	•	•	•	•	•	•	•	•	4	3	2	1	0
0	0										0	MSK	PR2	PR1	PR0

Figure 28. Timer/DMA Control Register Formats

15	14	•	•	•	•	•	•	7	6	5	4	3	2	1	0
0	0							0	SFNM	C	LTM	MSK	PR2	PR1	PR0

Figure 29. INT0/INT1 Control Register Formats

15	14	•	•	•	•	•	•	•	•	5	4	3	2	1	0
0	0									0	LTM	MSK	PR2	PR1	PR0

Figure 30. INT2/INT3 Control Register Formats

POLL AND POLL STATUS REGISTERS

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80188 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80188 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control regisers before enable interrupts.

Slave Mode External Interface

The configuration of the 80188 with respect to an external 8259A master is shown in Figure 33. The INT0 (pin 45) input is used as the 80188 CPU interrupt input. INT3 (pin 41) functions as an output to send the 80188 slave-interrupt-request to one of the 8 master-PIC-inputs.

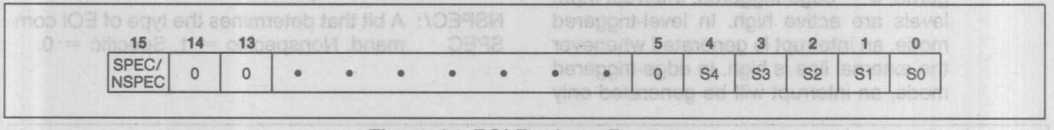


Figure 31. EOI Register Format

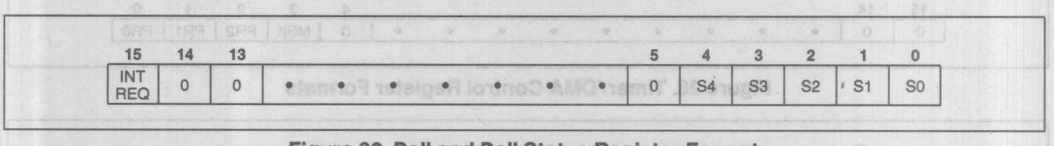


Figure 32. Poll and Poll Status Register Format

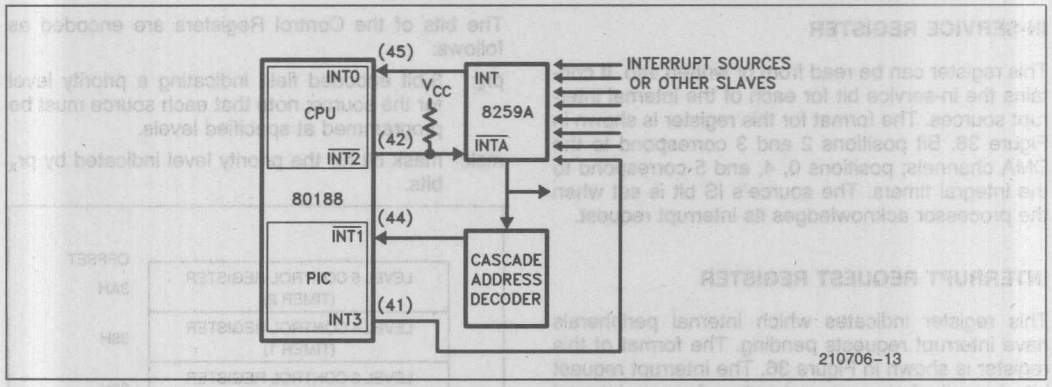


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0–2). Slave 8259As do this internally. Because of pin limitations, the 80188 slave address will have to be decoded externally. $\overline{INT1}$ (pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ (pin 42) is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

END-OF-INTERRUPT REGISTER

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

IN-SERVICE REGISTER

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

INTERRUPT REQUEST REGISTER

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write, all other bits are read only.

MASK REGISTER

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

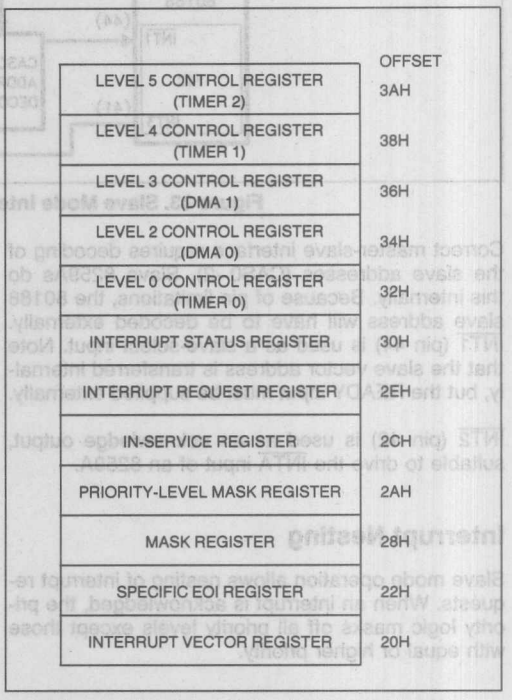


Figure 34. Interrupt Controller Registers (Slave Mode)

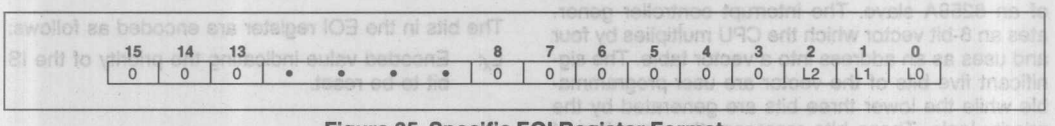


Figure 35. Specific EOI Register Format

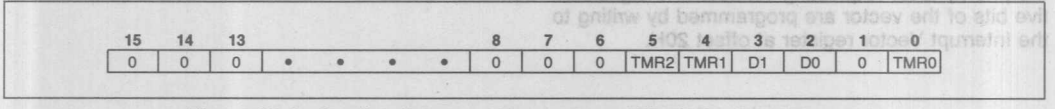


Figure 36. In-Service, Interrupt Request, and Mask Register Format

INTERRUPT VECTOR REGISTER

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

PRIORITY-LEVEL MASK REGISTER

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

INTERRUPT STATUS REGISTER

This register is defined as in master mode except that DHLT is not implemented. (See Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

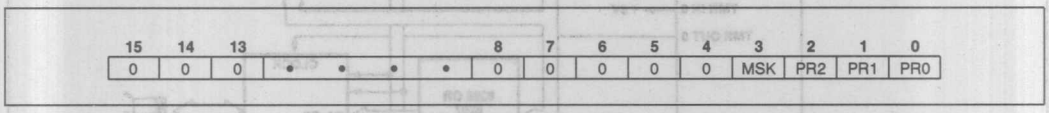


Figure 37. Control Word Format

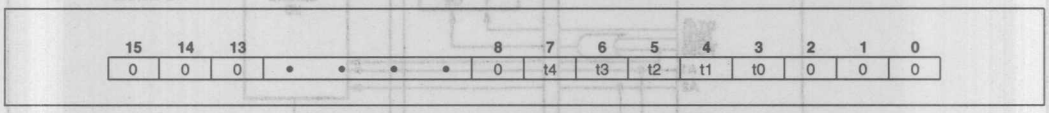


Figure 38. Interrupt Vector Register Format

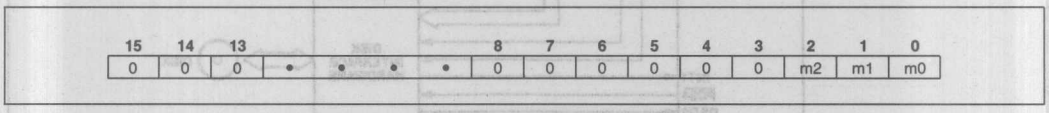


Figure 39. Priority Level Mask Register

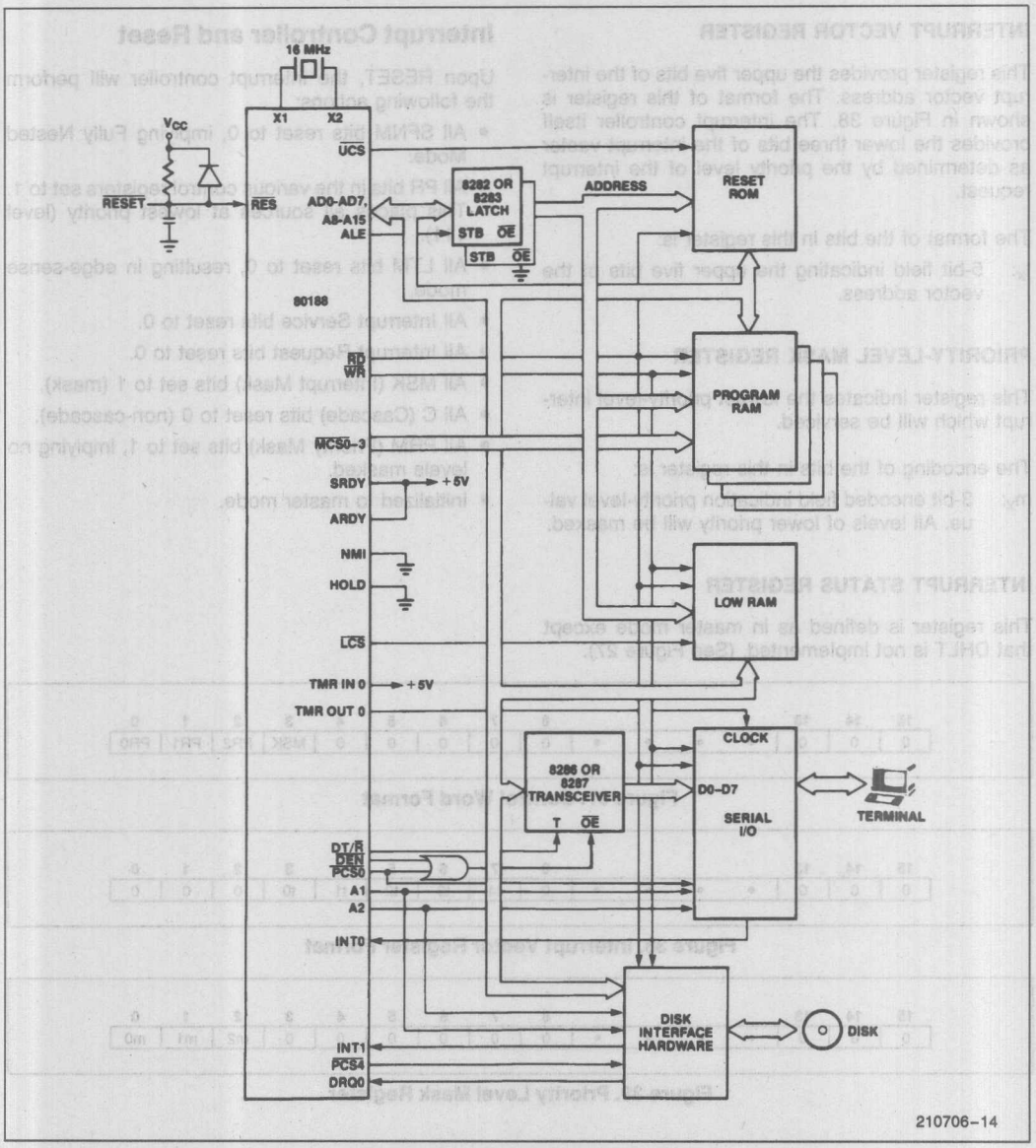


Figure 40. Typical 80188 Computer

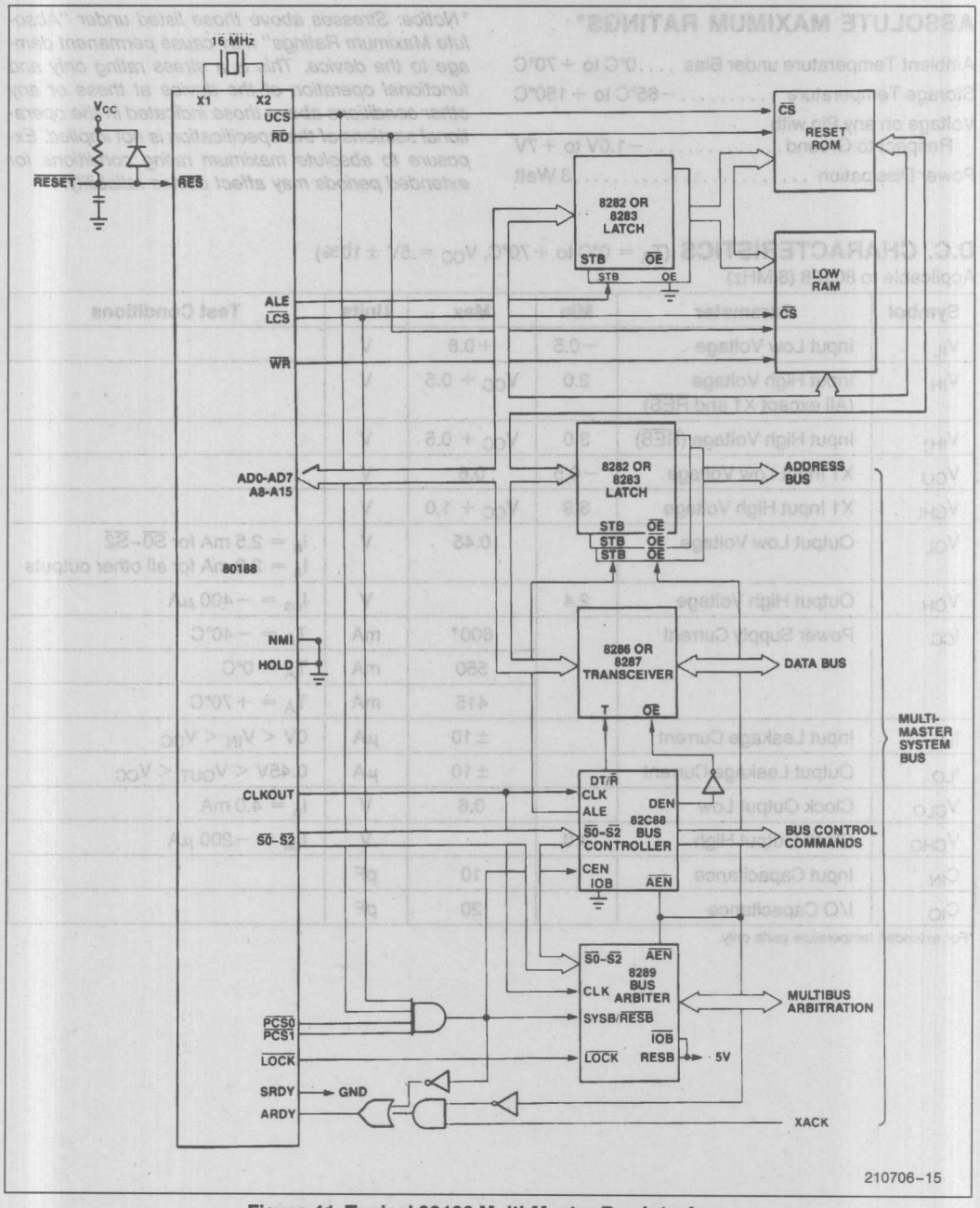


Figure 41. Typical 80188 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Voltage on any Pin with
Respect to Ground.....−1.0V to +7V
Power Dissipation .....3 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

Applicable to 80188 (8 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{CLI}	X1 Input Low Voltage	−0.5	0.6	V	
V_{CHI}	X1 Input High Voltage	3.9	V_{CC} + 1.0	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

80188 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted

Symbol	Parameter	80188 (8 MHz)		Units	Test Conditions
		Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		ns	
T_{CLDX}	Data in Hold (A/D)	10		ns	
T_{ARYHCH}	Asynchronous Ready (ARDY) active setup time(1)	20		ns	
T_{ARYLCL}	ARDY inactive setup time	35		ns	
T_{CLARX}	ARDY hold time	15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time(2)	20		ns	
T_{CLSRY}	SRDY Transition Hold Time(2)	15		ns	
T_{HVCL}	HOLD Setup(1)	25		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TMR IN, Setup(1)	25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup(1)	25		ns	

80188 Master Interface Timing Responses

T_{CLAV}	Address Valid Delay	5	55	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 MHz
T_{CLAX}	Address Hold	10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	ns	
T_{CHCZ}	Command Lines Float Delay		45	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		ns	
T_{CHLH}	ALE Active Delay		35	ns	
T_{CHLL}	ALE Inactive Delay		35	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		ns	
T_{CLDV}	Data Valid Delay	10	44	ns	
T_{CLDOX}	Data Hold Time	10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	ns	
T_{CHCTV}	Control Active Delay 2	10	55	ns	
T_{CVCTX}	Control Inactive Delay	5	55	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	ns	

1. To guarantee recognition at next clock.
2. To guarantee proper operation.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS

(T_A = 0°C to +70°C, V_{CC} = 5V ±10%) (Continued)

80188 Master Interface Timing Responses (Continued)

Symbol	Parameter	80188 (8 MHz)		Units	Test Conditions
		Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		ns	
T_{AVLL}	Address Valid to ALE Low	$T_{CLCH}-25$		ns	
T_{CHSV}	Status Active Delay	10	55	ns	
T_{CLSH}	Status Inactive Delay	10	65	ns	
T_{CLTMV}	Timer Output Delay		60	ns	100 pF max
T_{CLRO}	Reset Delay		60	ns	
T_{CHQSV}	Queue Status Delay		35	ns	
T_{CHDX}	Status Hold Time	10		ns	
T_{AVCH}	Address Valid to Clock High	10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	ns	

80188 Chip-Select Timing Responses

T_{CLCSV}	Chip-Select Active Delay		66	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	ns	

80188 CLKIN Requirements

T_{CKIN}	CLKIN Period	62.5	250	ns	
T_{CKHL}	CLKIN Fall Time		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		ns	1.5V
T_{CHCK}	CLKIN High Time	25		ns	1.5V

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS (Continued)

(T_A = 0°C to +70°C, V_{CC} = 5V ±10%) (Continued)

80188 CLKOUT Timing (200 pF load)

Symbol	Parameter	80188 (8 MHz)		Units	Test Conditions
		Min	Max		
T_{CICO}	CLKIN to CLKOUT Skew		50	ns	
T_{CLCL}	CLKOUT Period	125	500	ns	
T_{CLCH}	CLKOUT Low Time	½ T_{CLCL} −7.5		ns	1.5V
T_{CHCL}	CLKOUT High Time	½ T_{CLCL} −7.5		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15	ns	3.5 to 1.0V

Explanation of the AC Symbols

Each timing symbol has from 5 to 7 characters. The first character is always a "T" (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

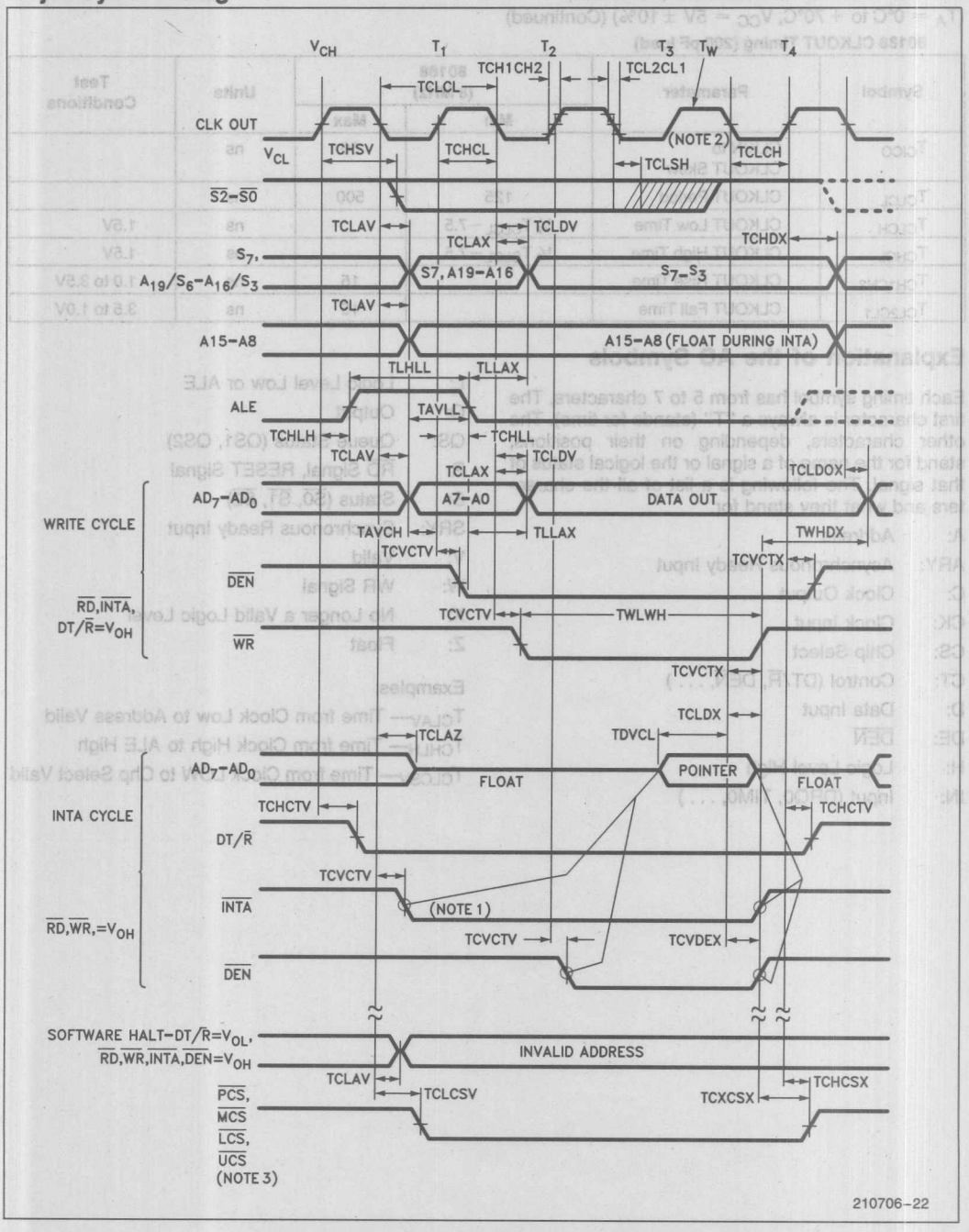
A: Address
ARY: Asynchronous Ready Input
C: Clock Output
CK: Clock Input
CS: Chip Select
CT: Control (DT/$\overline{R}$, $\overline{DEN}$, . . .)
D: Data Input
DE: $\overline{DEN}$
H: Logic Level High
IN: Input (DRQ0, TIM0, . . .)
L: Logic Level Low or ALE
O: Output
QS: Queue Status (QS1, QS2)
R: $\overline{RD}$ Signal, RESET Signal
S: Status ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$)
SRY: Synchronous Ready Input
V: Valid
W: WR Signal
X: No Longer a Valid Logic Level
Z: Float

Examples:

T_{CLAV}— Time from Clock Low to Address Valid
T_{CHLH}— Time from Clock High to ALE High
T_{CLCSV}— Time from Clock LOW to Chp Select Valid

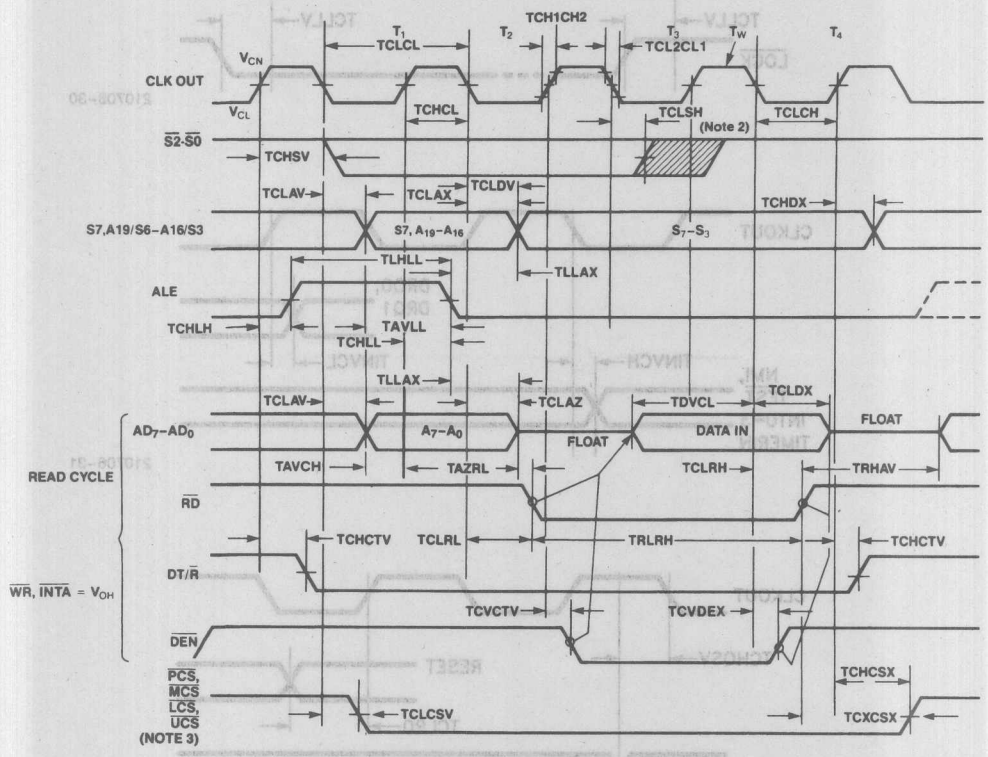
WAVEFORMS

Major Cycle Timing



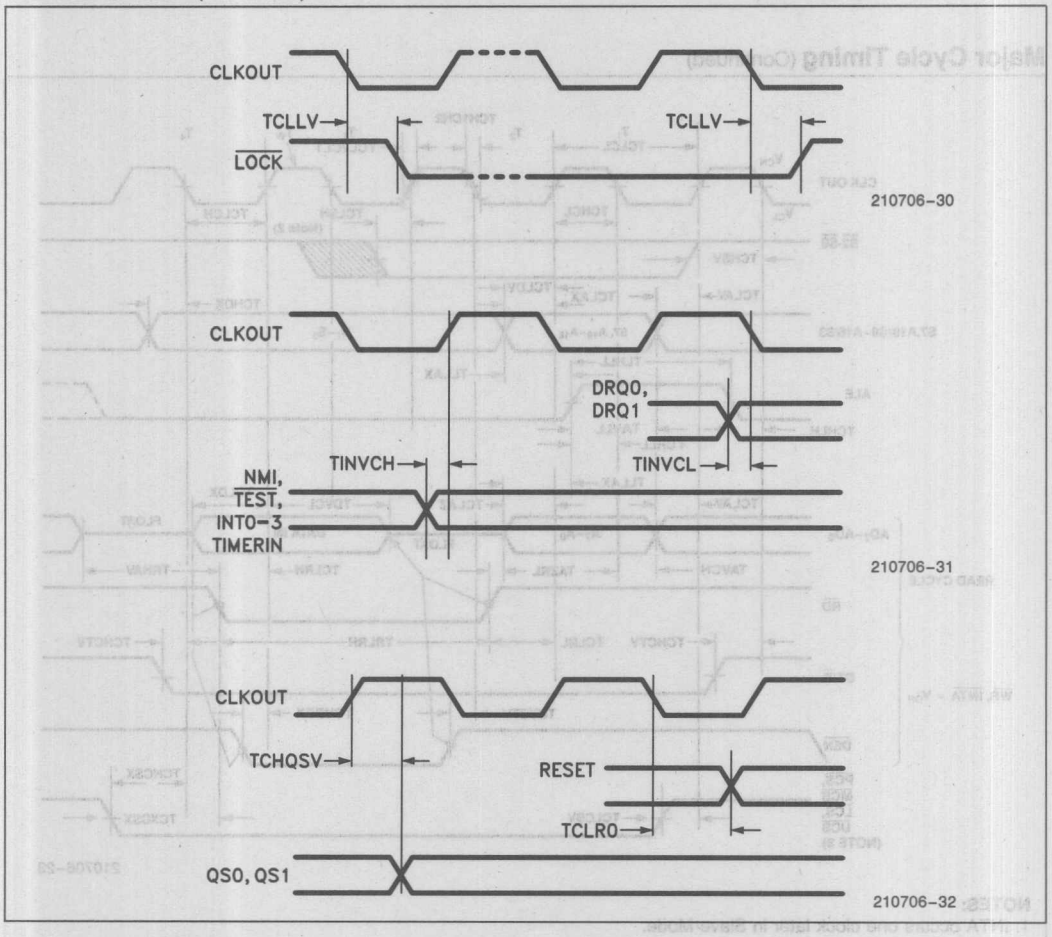
WAVEFORMS (Continued)

Major Cycle Timing (Continued)



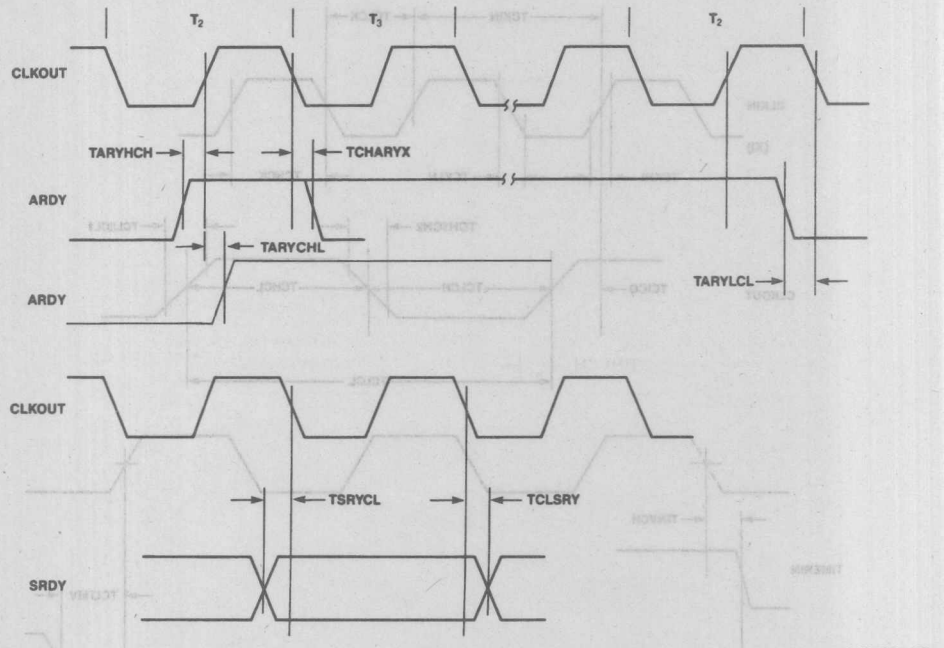
NOTES:
1. INTA occurs one clock later in Slave Mode.
2. Status inactive just prior to T_4.
3. Latched A1 and A2 have same timings as $\overline{PCS5}$ and $\overline{PCS6}$.

WAVEFORMS (Continued)

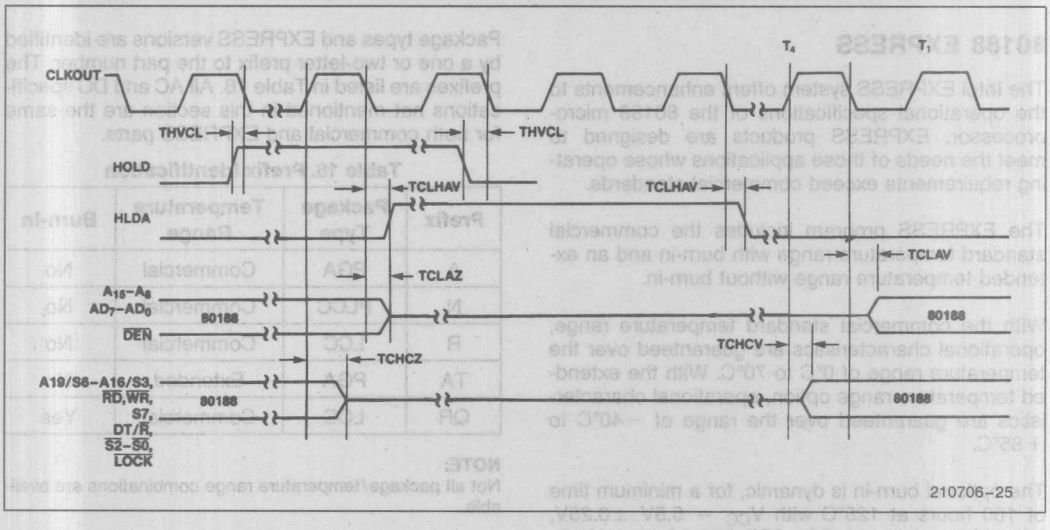


WAVEFORMS (Continued)

READY TIMING

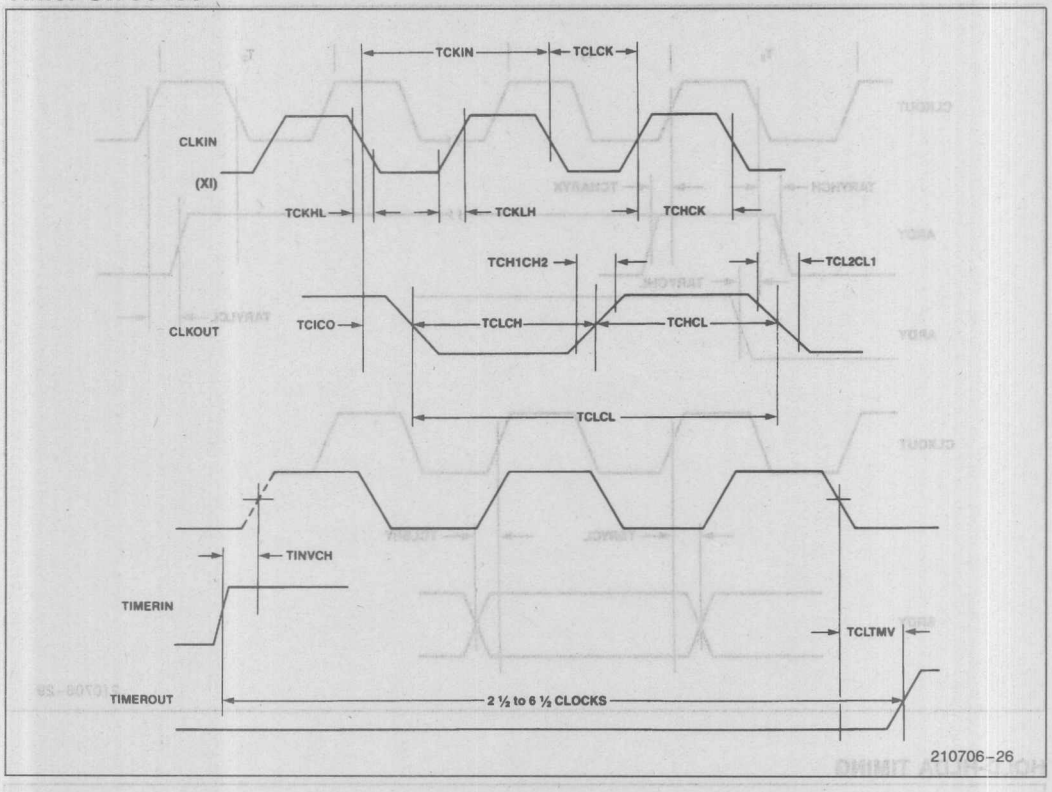


HOLD-HLDA TIMING



WAVEFORMS (Continued)

Timer On 80188



80188 EXPRESS

The Intel EXPRESS system offers enhancements to the operational specifications of the 80188 microprocessor. EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in and an extended temperature range without burn-in.

With the commercial standard temperature range, operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.25V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one or two-letter prefix to the part number. The prefixes are listed in Table 16. All AC and DC specifications not mentioned in this section are the same for both commercial and EXPRESS parts.

Table 16. Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
A	PGA	Commercial	No
N	PLCC	Commercial	No
R	LCC	Commercial	No
TA	PGA	Extended	No
QR	LCC	Commercial	Yes

NOTE:
Not all package/temperature range combinations are available.

80188 EXECUTION TIMINGS

Since the bus interface unit and execution unit operate independently, a determination of 80188 program execution timing must consider the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.

All instructions which involve memory accesses can also require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

The 80188 8-bit BIU is noticeably limited in its performance relative to the execution unit. A sufficient number of prefetched bytes may not reside in the prefetch queue much of the time. Therefore, actual program execution time may be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to register/memory	1000100w	mod reg r/m			2/12*	
Register/memory to register	1000101w	mod reg r/m			2/9*	
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1	12/13*	8/16-bit
Immediate to register	1011w reg	data	data if w=1		3/4	8/16-bit
Memory to accumulator	1010000w	addr-low	addr-high		8*	
Accumulator to memory	1010001w	addr-low	addr-high		9*	
Register/memory to segment register	10001110	mod 0 reg r/m			2/13	
Segment register to register/memory	10001100	mod 0 reg r/m			2/15	
PUSH = Push:						
Memory	11111111	mod 1 1 0 r/m			20	
Register	01010 reg				14	
Segment register	000reg110				13	
Immediate	011010s0	data	data if s=0		14	
PUSHA = Push All	01100000				68	
POP = Pop:						
Memory	10001111	mod 0 0 0 r/m			24	
Register	01011 reg				14	
Segment register	000reg111	(reg≠01)			12	
POPA = Pop All	01100001				83	
XCHG = Exchange:						
Register/memory with register	1000011w	mod reg r/m			4/17*	
Register with accumulator	10010 reg				3	
IN = Input from:						
Fixed port	1110010w	port			10*	
Variable port	1110110w				8*	
OUT = Output to:						
Fixed port	1110011w	port			9*	
Variable port	1110111w				7*	
XLAT = Translate byte to AL	11010111				15	
LEA = Load EA to register	10001101	mod reg r/m			6	
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	10011111				2	
SAHF = Store AH into flags	10011110				3	
PUSHF = Push flags	10011100				13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
POPF = Pop flags	10011101				12	
SEGMENT = **Segment Override:**						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = **Add:**						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = **Add with carry:**						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = **Increment:**						
Register/memory	1111111w	mod 000 r/m			3/15*	
Register	01000 reg				3	
SUB = **Subtract:**						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = **Subtract with borrow:**						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = **Decrement:**						
Register/memory	1111111w	mod 001 r/m			3/15*	
Register	01001 reg				3	
CMP = **Compare:**						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10*	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
ARITHMETIC (Continued)			
MUL = Multiply (unsigned):	1111011w | mod 100 r/m		
Register-Byte		26–28	
Register-Word		35–37	
Memory-Byte		32–34	
Memory-Word		41–43*	
IMUL = Integer multiply (signed):	1111011w | mod 101 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43*	
IMUL = Integer Immediate multiply (signed)	011010s1 | mod reg r/m | data | data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1111011w | mod 110 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44*	
IDIV = Integer divide (signed):	1111011w | mod 111 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67*	
AAM = ASCII adjust for multiply	11010100 | 00001010	19	
AAD = ASCII adjust for divide	11010101 | 00001010	15	
CBW = Convert byte to word	10011000	2	
CWD = Convert word to double word	10011001	4	
LOGIC			
Shift/Rotate Instructions:			
Register/Memory by 1	1101000w | mod TTT r/m	2/15	
Register/Memory by CL	1101001w | mod TTT r/m	5+n/17+n	
Register/Memory by Count	1100000w | mod TTT r/m | count	5+n/17+n	

TTT	Instruction
000	ROL
001	ROR
010	RCL
011	RCR
100	SHL/SAL
101	SHR
111	SAR

AND = And:					
Reg/memory and register to either	001000dw	mod reg r/m		3/10*	
Immediate to register/memory	1000000w	mod 100 r/m	data | data if w=1	4/16*	
Immediate to accumulator	0010010w	data	data if w=1	3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
LOGIC (Continued)			
TEST = And function to flags, no result:			
Register/memory and register	1000010w | mod reg r/m	3/10*	
Immediate data and register/memory	1111011w | mod 000 r/m | data | data if w=1	4/10*	
Immediate data and accumulator	1010100w | data | data if w=1	3/4	8/16-bit
OR = Or:			
Reg/memory and register to either	000010dw | mod reg r/m	3/10*	
Immediate to register/memory	1000000w | mod 001 r/m | data | data if w=1	4/16*	
Immediate to accumulator	0000110w | data | data if w=1	3/4	8/16-bit
XOR = Exclusive or:			
Reg/memory and register to either	001100dw | mod reg r/m	3/10*	
Immediate to register/memory	1000000w | mod 110 r/m | data | data if w=1	4/16*	
Immediate to accumulator	0011010w | data | data if w=1	3/4	8/16-bit
NOT = Invert register/memory	1111011w | mod 010 r/m	3/10*	
STRING MANIPULATION:			
MOVS = Move byte/word	1010010w	14*	
CMPS = Compare byte/word	1010011w	22*	
SCAS = Scan byte/word	1010111w	15*	
LODS = Load byte/wd to AL/AX	1010110w	12*	
STOS = Store byte/wd from AL/AX	1010101w	10*	
INS = Input byte/wd from DX port	0110110w	14	
OUTS = Output byte/wd to DX port	0110111w	14	
Repeated by count in CX (REP/REPE/REPZ/REPNE/REPNZ)			
MOVS = Move string	11110010 | 1010010w	8+8n*	
CMPS = Compare string	1111001z | 1010011w	5+22n*	
SCAS = Scan string	1111001z | 1010111w	5+15n*	
LODS = Load string	11110010 | 1010110w	6+11n*	
STOS = Store string	11110010 | 1010101w	6+9n*	
INS = Input string	11110010 | 0110110w	8+8n*	
OUTS = Output string	11110010 | 0110111w	8+8n*	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	19	
Register/memory indirect within segment	11111111 | mod 0 1 0 r/m	17/27	
Direct intersegment	10011010 | segment offset segment selector	31	
Indirect intersegment	11111111 | mod 0 1 1 r/m | (mod ≠ 11)	54	
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 1 0 0 r/m	11/21	
Direct intersegment	11101010 | segment offset segment selector	14	
Indirect intersegment	11111111 | mod 1 0 1 r/m | (mod ≠ 11)	34	
RET = Return from CALL:			
Within segment	11000011	20	
Within seg adding immed to SP	11000010 | data-low | data-high	22	
Intersegment	11001011	30	
Intersegment adding immediate to SP	11001010 | data-low | data-high	33	
JE/JZ = Jump on equal/zero	01110100 | disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100 | disp	4/13	
JLE/JNG = Jump on less or equal/not greater	01111110 | disp	4/13	
JB/JNAE = Jump on below/not above or equal	01110010 | disp	4/13	
JBE/JNA = Jump on below or equal/not above	01110110 | disp	4/13	
JP/JPE = Jump on parity/parity even	01111010 | disp	4/13	
JO = Jump on overflow	01110000 | disp	4/13	
JS = Jump on sign	01111000 | disp	4/13	
JNE/JNZ = Jump on not equal/not zero	01110101 | disp	4/13	
JNL/JGE = Jump on not less/greater or equal	01111101 | disp	4/13	
JNLE/JG = Jump on not less or equal/greater	01111111 | disp	4/13	
JNB/JAE = Jump on not below/above or equal	01110011 | disp	4/13	
JNBE/JA = Jump on not below or equal/above	01110111 | disp	4/13	
JNP/JPO = Jump on not par/par odd	01111011 | disp	4/13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = **Interrupt:**						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	
PROCESSOR CONTROL						
CLC = Clear carry	11111000				2	
CMC = Complement carry	11110101				2	
STC = Set carry	11111001				2	
CLD = Clear direction	11111100				2	
STD = Set direction	11111101				2	
CLI = Clear interrupt	11111010				2	
STI = Set interrupt	11111011				2	
HLT = Halt	11110100				2	
WAIT = Wait	10011011				6	if $\overline{TEST}$ = 0
LOCK = Bus lock prefix	11110000				2	
ESC = Processor Extension Escape	11011TTT	mod LLL r/m			6	
	(TTT LLL are opcode to processor extension)					
NOP = No Operation	10010000				3	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field

if mod = 00 then DISP = 0*, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP*

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

REVISION HISTORY

The sections significantly revised since version -008 are:

Pin Description Table	Noted $\overline{RES}$ to be low more than 4 clocks. Connections to X1 and X2 clarified.
DMA Control Bit Descriptions	Moved and clarified note concerning TC condition for ST/$\overline{STOP}$ clearing during unsynchronized transfers.
Interrupt Controller, etc.	Renamed iRMX Mode to Slave Mode.
Interrupt Request Register	Noted that D0 and D1 are read/write, others read-only.
Execution Timings	Clarified effect of bus width.
A.C. Characteristics	10 MHz 80188 no longer offered.

The sections significantly revised since version -009 are:

Pin Description Table	Various descriptions rewritten for clarity.
Interrupt Vector Table	Redrawn for clarity.
A.C. Characteristics	Added reminder that T_{SRYCL} and T_{CLSRY} must be met.
Explanation of the A.C. Symbols	New section.
Major Cycle Timing Waveforms	T_{CLRO} indicated.

80C188
CHMOS HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Operation Modes Include:**
  - **Enhanced Mode Which Has**
    - **DRAM Refresh**
    - **Power-Save Mode**
  - **Compatible Mode**
    - **NMOS 80188 Pin for Pin Replacement for Non-Numerics Applications**
- **Integrated Feature Set**
  - **Enhanced 80C86/C88 CPU**
  - **Clock Generator**
  - **2 Independent DMA Channels**
  - **Programmable Interrupt Controller**
  - **3 Programmable 16-Bit Timers**
  - **Dynamic RAM Refresh Control Unit**
  - **Programmable Memory and Peripheral Chip Select Logic**
  - **Programmable Wait State Generator**
  - **Local Bus Controller**
  - **Power Save Mode**
  - **System-Level Testing Support (High Impedance Test Mode)**
- **Available in 16 MHz (80C188-16), 12.5 MHz (80C188-12) and 10 MHz (80C188) Versions**
- **Direct Addressing Capability to 1 MByte Memory and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086/8088 Software and Also Has 10 Additional Instructions over 8086/8088**
- **Complete System Development Support**
  - **All 8088 and NMOS 80188 Software Development Tools Can Be Used for 80C186 System Development**
    - **ASM86 Assembler, PL/M-86, Pascal-86, Fortran-86, C-86 and System Utilities**
    - **In-Circuit-Emulator (ICE™-188)**
- **Available in 68 Pin:**
  - **Plastic Leaded Chip Carrier (PLCC)**
  - **Ceramic Pin Grid Array (PGA)**
  - **Ceramic Leadless Chip Carrier (JEDEC A Package)**

  (See Packaging Outlines and Dimensions, Order Number 231369)
- **Available in EXPRESS:**
  - **Standard Temperature with Burn-In**
  - **Extended Temperature Range (−40°C to +85°C)**

The Intel 80C188 is a CHMOS high integration microprocessor. It has features which are new to the 80186 family which include a DRAM refresh control unit, power-save mode. When used in "compatible" mode, the 80C188 is 100% pin-for-pin compatible with the NMOS 80188 (except for 8087 applications). The "enhanced" mode of operation allows the full feature set of the 80C188 to be used. The 80C188 is upward compatible with 8086 and 8088 software and fully compatible with 80186 and 80188 software, except for numerics applications.

October 1988
Order Number: 270432-002

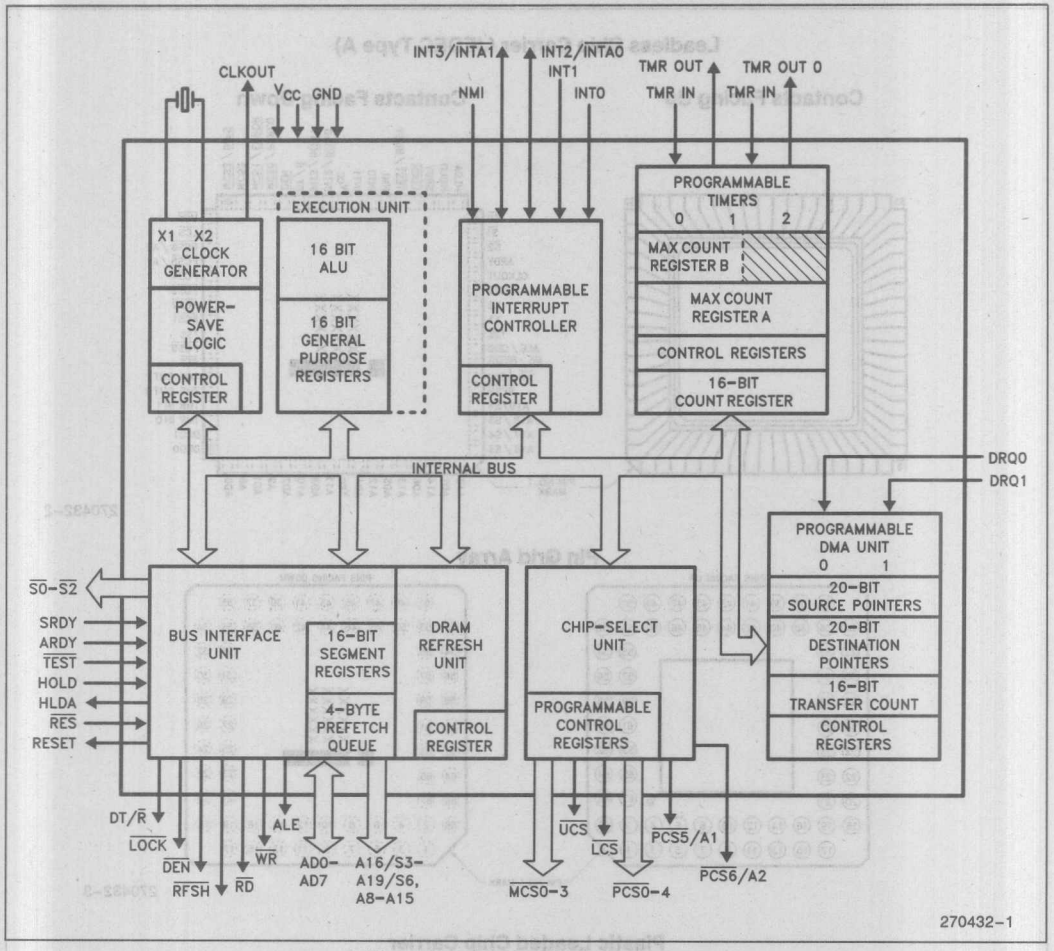


Figure 1. 80C188 Block Diagram

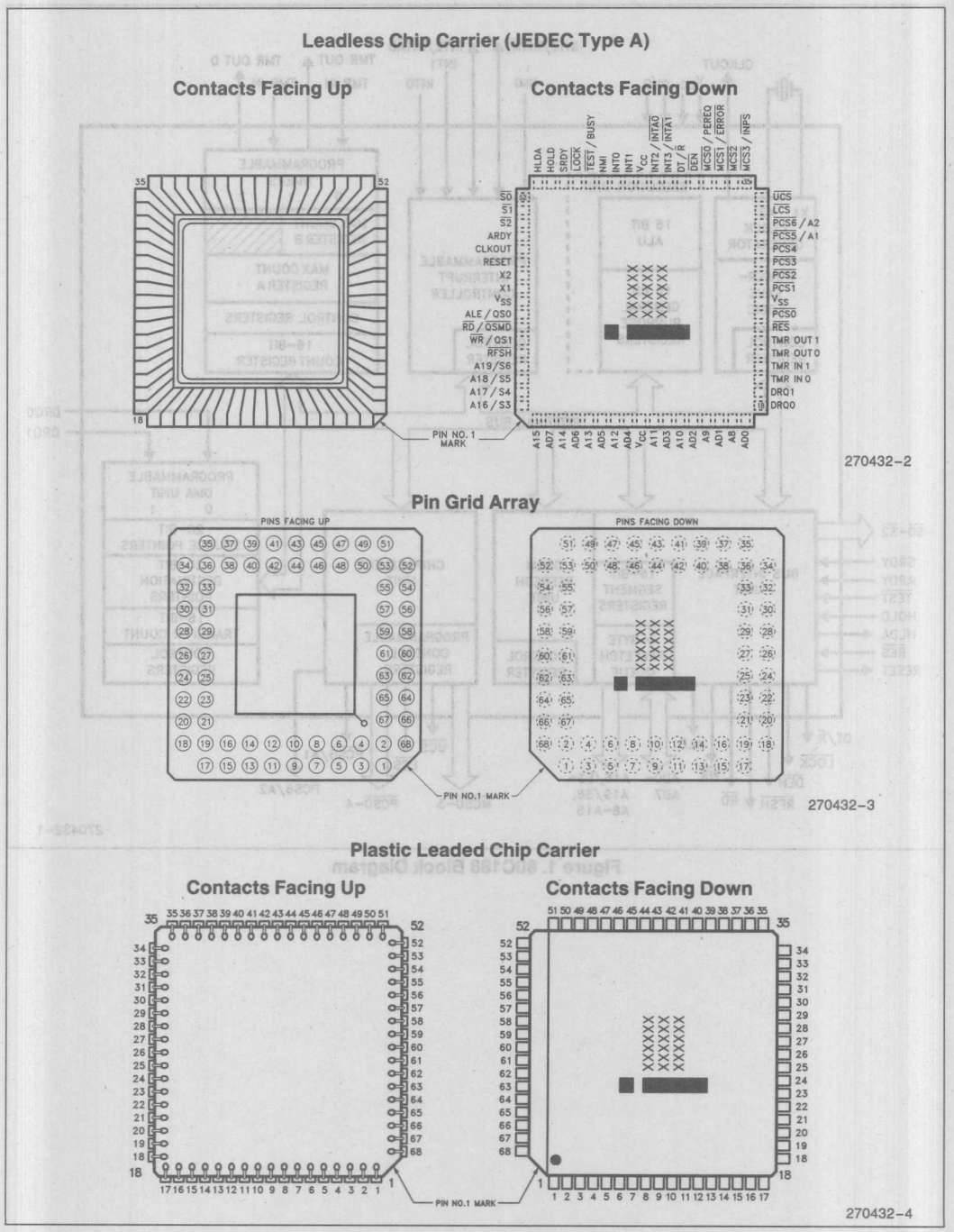


Figure 2. 80C188 Pinout Diagrams

Table 1. 80C188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	9 43	I I	System Power: +5 volt power supply.
V_{SS}	26 60	I I	System Ground.
RESET	57	O	Reset Output indicates that the 80C188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal. Reset goes inactive 2 clockout periods after $\overline{RES}$ goes inactive. When tied to the $\overline{TEST}$ pin, Reset forces the 80C188 into enhanced mode. Reset Output is not floated during bus hold.
X1 X2	59 58	I O	Crystal Inputs X1 and X2 provide external connections for a fundamental mode or third overtone parallel resonant crystal for the internal oscillator. X1 can connect to an external clock instead of a crystal. In this case, minimize the capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT is active during reset and bus hold.
$\overline{RES}$	24	I	An active $\overline{RES}$ causes the 80C188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80C188 clock. The 80C188 begins fetching instructions approximately 6½ clock cycles after $\overline{RES}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{RES}$ held LOW. $\overline{RES}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network.
$\overline{TEST}$	47	I	The $\overline{TEST}$ pin is sampled during and after reset to determine whether the 80C188 is to enter Compatible or Enhanced Mode. Enhanced Mode requires $\overline{TEST}$ to be HIGH on the rising edge of $\overline{RES}$ and LOW four clocks later. Any other combination will place the 80C188 in Compatible Mode. A weak internal pullup (750Ω ±20%) insures a HIGH state when the pin is not driven. This pin is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when WAIT execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled every five clocks until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80C188 is waiting for $\overline{TEST}$, interrupts will be serviced.
TMR IN 0 TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized. Timer Inputs must be tied HIGH when not being used as clock or retrigger inputs.
TMR OUT 0 TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected. These outputs are not floated during a bus hold.

Table 1. 80C188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
DRQ0 DRQ1	18 19	I	DMA Request is asserted HIGH by an external device when it is ready for DMA Channel 0 or 1 to perform a transfer. These signals are level-triggered and internally synchronized.
NMI	46	I	The Non-Maskable Interrupt input causes a Type 2 interrupt. An NMI transition from LOW to HIGH is latched and synchronized internally, and initiates the interrupt at the next instruction boundary. NMI must be asserted for at least one clock. The Non-Maskable Interrupt cannot be avoided by programming.
INT0 INT1 INT2/$\overline{\text{INTA0}}$ INT3/$\overline{\text{INTA1}}$	45 44 42 41	I I I/O I/O	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
A19/S6 A18/S5 A17/S4 A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) indicate the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: During T_2, T_3, T_W, and T_4, the S6 pin is LOW to indicate a CPU-initiated bus cycle or HIGH to indicate a DMA-initiated bus cycle. During the same T-states, S3, S4, and S5 are always LOW. These outputs are floated during a bus hold or reset.
A15 A14 A13 A12 A11 A10 A9 A8	1 3 5 7 10 12 14 16	O O O O O O O O	Address-Only Bus (15–8) contains valid addresses from T_1–T_4. The bus is active high. These outputs are floated during a bus hold or reset.
AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0	2 4 6 8 11 13 15 17	I/O I/O I/O I/O I/O I/O I/O I/O	Address/Data Bus (7–0) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active high. These pins are floated during a bus hold or reset.
$\overline{\text{RFSH}}$	64	O	In compatible mode, $\overline{\text{RFSH}}$ is HIGH. In enhanced mode, $\overline{\text{RFSH}}$ is asserted LOW to signify a refresh bus cycle. The $\overline{\text{RFSH}}$ output pin floats during bus hold or reset, regardless of operating mode.

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80C188 to latch the address. ALE is active HIGH, with addresses guaranteed valid on the trailing edge. ALE floats during reset, but not during bus hold.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. It is active LOW, and floats during bus hold or reset. When the 80C188 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First opcode byte fetched from the queue
1	1	Subsequent byte fetched from the queue
1	0	Empty the queue

$\overline{RD}$/$\overline{QSMD}$	62	I/O	Read Strobe is an active LOW signal which indicates that the 80C188 is performing a memory or I/O read cycle. It is guaranteed not to go LOW before the A/D bus is floated. An internal pull-up (750Ω ±20%) ensures that $\overline{RD}$/$\overline{QSMD}$ is HIGH during RESET. Following RESET the pin is sampled to determine whether the 80C188 is to provide ALE, $\overline{RD}$ and $\overline{WR}$, or queue status information. To enable Queue Status Mode, $\overline{RD}$ must be connected to GND. $\overline{RD}$ will float during bus hold.
ARDY	55	I	Asynchronous Ready informs the 80C188 that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT and is active HIGH. The falling edge of ARDY must be synchronized to the 80C188 clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.
SRDY	49	I	Synchronous Ready informs the 80C188 that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows a relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchronize the ARDY input signal. Connecting SRDY high will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the ARDY pin.

Table 1. 80C188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus. $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction immediately following the LOCK prefix. It remains active until the completion of that instruction. No instruction prefetching will occur while $\overline{LOCK}$ is asserted. $\overline{LOCK}$ floats during bus hold or reset.
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	52 53 54	O O O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information: **80C188 Bus Cycle Status Information** $\overline{S2}$ $\overline{S1}$ $\overline{S0}$ — Bus Cycle Initiated 0 0 0 — Interrupt Acknowledge 0 0 1 — Read I/O 0 1 0 — Write I/O 0 1 1 — Halt 1 0 0 — Instruction Fetch 1 0 1 — Read Data from Memory 1 1 0 — Write Data to Memory 1 1 1 — Passive (no bus cycle) The status pins float during HOLD/HLDA. $\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
HOLD HLDA	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. The 80C188 generates HLDA (HIGH) in response to a HOLD request. Simultaneous with the issuance of HLDA, the 80C188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80C188 will lower HLDA. When the 80C188 needs to run another bus cycle, it will again drive the local bus and control lines. In Enhanced Mode, HLDA will go low when a DRAM refresh cycle is pending in the 80C188 and an external bus master has control of the bus. It will be up to the external master to relinquish the bus by lowering HOLD so that the 80C188 may execute the refresh cycle. Lowering HOLD for one clock and returning HIGH will ensure only one refresh cycle to the external master. HLDA will immediately go active after the refresh cycle has taken place.
$\overline{UCS}$	34	I/O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. $\overline{UCS}$ does not float during bus hold. The address range activating $\overline{UCS}$ is software programmable. $\overline{UCS}$ and $\overline{LCS}$ are sampled upon the rising edge of $\overline{RES}$. If both pins are held low, the 80C188 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{UCS}$ has a weak internal pullup (750Ω ±20%) that is active during RESET to ensure that the 80C188 does not enter the ONCE mode inadvertently.

Table 1. 80C188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{LCS}$	33	I/O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. $\overline{LCS}$ does not float during bus HOLD. The address range activating $\overline{LCS}$ is software programmable. $\overline{UCS}$ and $\overline{LCS}$ are sampled upon the rising edge of $\overline{RES}$. If both pins are held low, the 80C186 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{LCS}$ has a weak internal pullup (750Ω ±20%) that is active only during RESET to ensure that the 80C188 does not enter ONCE Mode inadvertently.
$\overline{MCS0}$ $\overline{MCS1}$ $\overline{MCS2}$ $\overline{MCS3}$	38 37 36 35	O O O O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines do not float during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$ $\overline{PCS2}$ $\overline{PCS3}$ $\overline{PCS4}$	25 27 28 29 30	O O O O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K I/O space or 1 Mbyte memory space). These lines do not float during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1 rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. $\overline{PCS5}$/A1 does not float during bus hold.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2 rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. $\overline{PCS6}$/A2 does not float during bus hold.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the 80C188. When HIGH the 80C188 places write data on the data bus. DT/$\overline{R}$ floats during a bus hold or RESET.
$\overline{DEN}$	39	O	Data Enable is provided as a data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state. $\overline{DEN}$ floats during bus hold or RESET.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80C188. The 80C188 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip. The 80C188 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the 8086/8088 instruction set.

The 80C188 has two major modes of operation, Compatible and Enhanced. In Compatible Mode the 80C188 is completely compatible with NMOS 80188, with the exception of 8087 support. The Enhanced mode adds two new features to the system design. These are Power-Save control and Dynamic RAM refresh.

80C188 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80188 families all contain the same basic set of registers, instructions, and addressing modes. The 80C188 processor is upward compatible with the 8086 and 8088 CPUs.

Register Set

The 80C188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used for arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80C188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80C186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

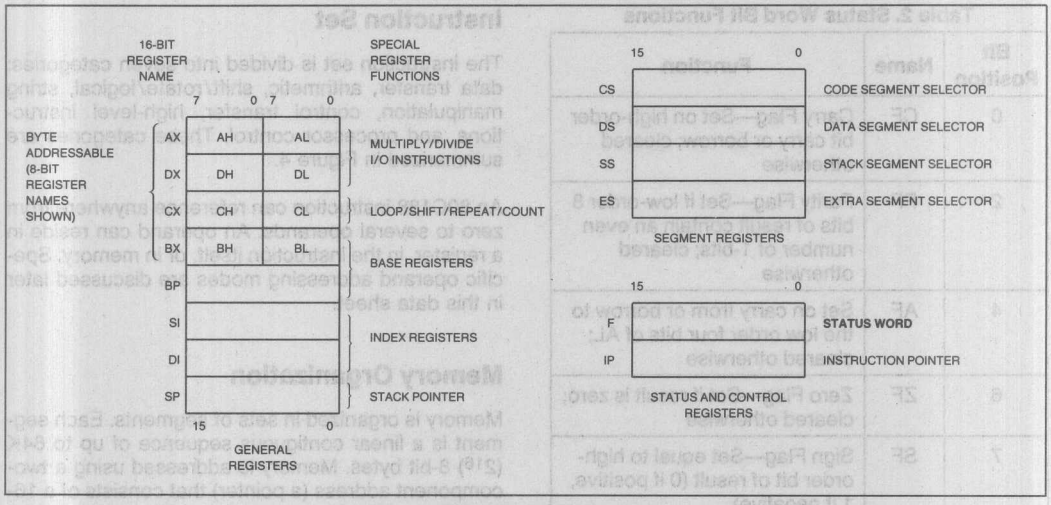


Figure 3a. 80C188 Register Set

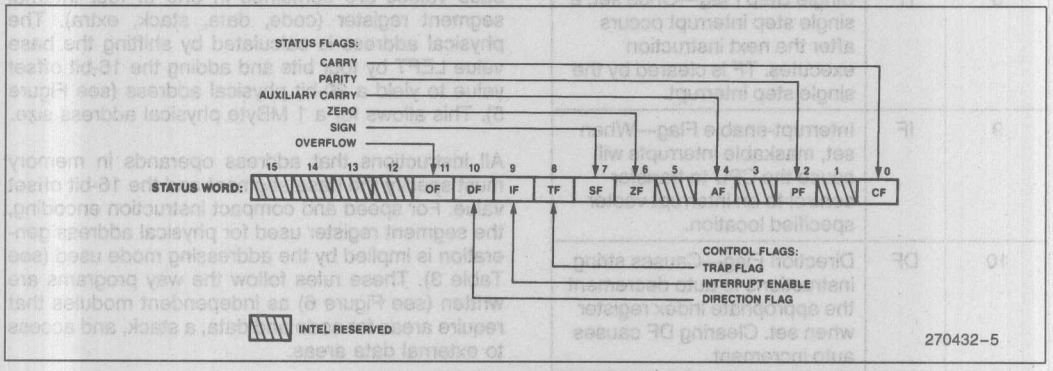


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80C188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80C188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80C188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

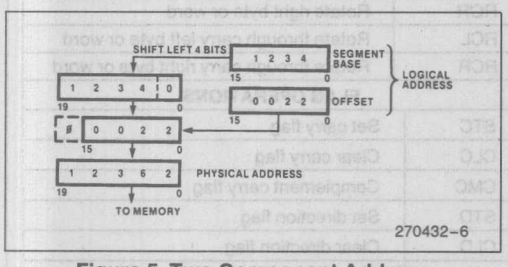


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

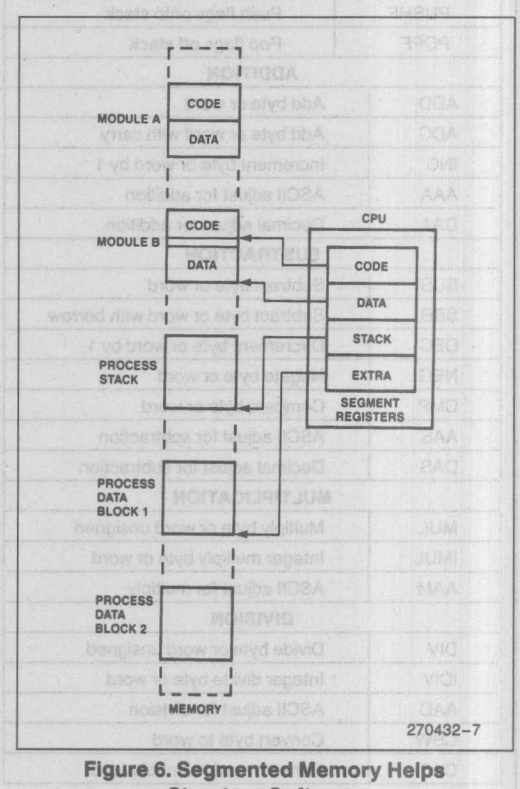


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80C188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80C188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.

In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80C188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by attempted execution of an ESC instruction, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding

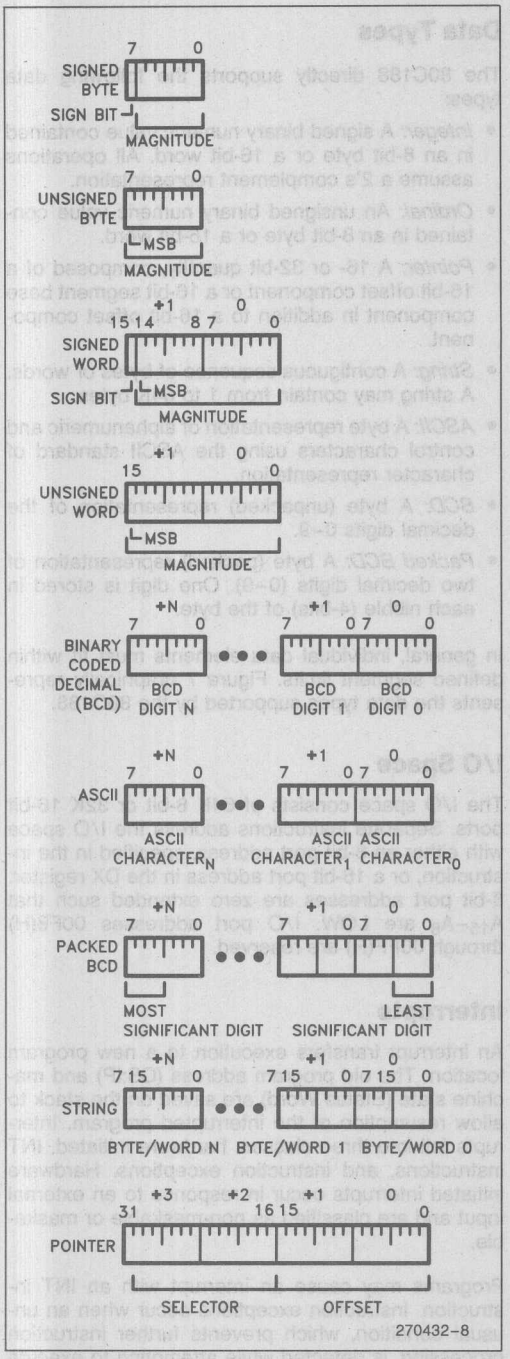


Figure 7. 80C188 Supported Data Types

the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80C188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80C186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80C188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80C188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80C188 Interrupt Vectors

Interrupt Name	Vector Type	Vector Address	Default Priority	Related Instructions	Applicable Notes
Divide Error Exception	0	00H	1	DIV, IDIV	1
Single Step Interrupt	1	04H	1A	All	2
Non-Maskable Interrupt (NMI)	2	08H	1	All	
Breakpoint Interrupt	3	0CH	1	INT	1
INT0 Detected Overflow Exception	4	10H	1	INTO	1
Array Bounds Exception	5	14H	1	BOUND	1
Unused-Opcode Exception	6	18H	1	Undefined Opcodes	1
ESC Opcode Exception	7	1CH	1	ESC Opcodes	1, 3
Timer 0 Interrupt	8	20H	2A		4
Timer 1 Interrupt	18	48H	2B		4
Timer 2 Interrupt	19	4CH	2C		4
Reserved	9	24H	3		
DMA 0 Interrupt	10	28H	4		
DMA 1 Interrupt	11	2CH	5		
INT0 Interrupt	12	30H	6		
INT1 Interrupt	13	34H	7		
INT2 Interrupt	14	38H	8		
INT3 Interrupt	15	3CH	9		
Reserved	16, 17	40H, 44H			
Reserved	20–31	50H . . . 7CH			

NOTES:
Default priorities for the interrupt sources are used only if the user does not program each source to a unique priority level.
1. Generated as a result of an instruction execution.
2. Performed in same manner as 8088.
3. An ESC opcode will cause a trap regardless of the 80C188 operating mode. The 80C188 is not directly compatible with the 80188 in this respect. The instruction set of a numerics coprocessor cannot be executed.
4. All three timers constitute one source of request to the interrupt controller. As such, they share the same priority level with respect to other interrupt sources. However, the timers have a defined priority order among themselves (2A > 2B > 2C).

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). The 80C188 does not check an escape opcode trap bit as does the 80C186. On the 80C188, ESC traps occcur in both compatible and enhanced operating modes. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

NOTE:
Unlike the 80188, all numerics coprocessor opcodes cause a trap. The 80C188 does not support the numerics interface.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80C188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80C188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80C188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80C188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80C188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80C188 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80C188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80C188 COMPARED TO THE 80C186

The 80C188 is an 8-bit processor designed based on the 80C186 internal structure. Most internal functions of the 80C188 are identical to the equivalent 80C186 functions. The 80C188 handles the external bus the same way the 80C186 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. The processors will look the same to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions except numerics instructions have the same end result. Internally, there are four differences between the 80C188 and the 80C186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80C188, whereas the 80C186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80C188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80C186 waits until a 2-byte space is available.
- The internal execution time of an instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the rate of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80C188 are being used, the queue has more time to fill and the execution proceeds more closely to the speed at which the execution unit will allow.
- The 80C188 does not have a numerics interface, since the 80C186 numerics interface inherently requires 16-bit communication with the numerics coprocessor.

The 80C188 and 80C186 are completely software compatible (except for numerics instructions) by virtue of their identical execution units. However, software that is system dependent may not be completely transferable.

The bus interface and associated control signals vary somewhat between the two processors. The pin assignments are nearly identical, with the following functional changes:

- A8–A15—These pins are only address outputs on the 80C188. These address lines are latched internally and remain valid throughout the bus cycle.
- $\overline{BHE}$ has no meaning on the 80C188. However, it was necessary to designate this pin the S7/$\overline{RFSH}$ pin in order to provide an indication of DRAM refresh bus cycles.

80C188 CLOCK GENERATOR

The 80C188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The 80C188 oscillator circuit is designed to be used either with a parallel resonant fundamental or third-overtone mode crystal, depending upon the frequency range of the application as shown in Figure 8c. This is used as the time base for the 80C188. The crystal frequency chosen should be twice the required processor frequency. Use of an LC or RC circuit is not recommended.

The output of the oscillator is not directly available outside the 80C188. The two recommended crystal configurations are shown in Figure 8a. When used in third-overtone mode the tank circuit shown in Figure 8b is recommended for stable operation. The sum of the stray capacitances and loading capacitors should equal the values shown. It is advisable to limit stray capacitance between the X1 and X2 pins to less than 10 pF. While a fundamental-mode circuit will require approximately 1 ms for start-up, the third-overtone arrangement may require 1 ms to 3 ms to stabilize.

Alternately, the oscillator may be driven from an external source as shown in Figure 8d. The configuration shown in Figure 8e is not recommended.

The following parameters should be used when choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	40Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pF max
C_1 (Load Capacitance):	20 pF ± 2 pF
Drive Level:	1 mW max

Clock Generator

The 80C188 clock generator provides the 50% duty cycle processor clock for the 80C188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80C188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80C188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used for full synchronization of a rising ARDY signal. A high-to-low transition on ARDY may be used as an indication of the not ready condition but it must be performed synchronously to CLKOUT **either** in the middle of T_2, T_3 or T_W, **or** at the falling edge of T_3 or T_W.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated. This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80C188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

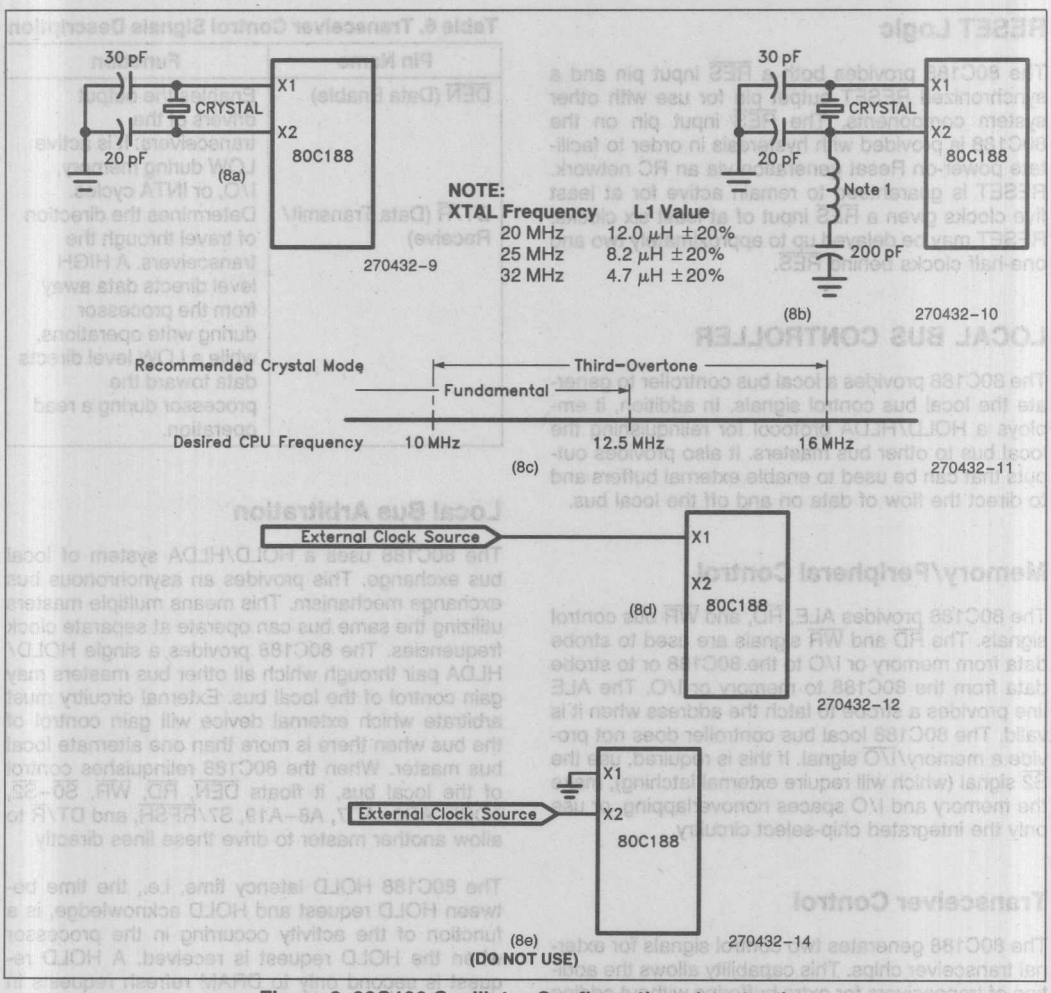


Figure 8. 80C188 Oscillator Configurations (see text)

RESET Logic

The 80C188 provides both a $\overline{RES}$ input pin and a synchronized RESET output pin for use with other system components. The $\overline{RES}$ input pin on the 80C188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to approximately two and one-half clocks behind $\overline{RES}$.

LOCAL BUS CONTROLLER

The 80C188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides outputs that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80C188 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory or I/O to the 80C188 or to strobe data from the 80C188 to memory or I/O. The ALE line provides a strobe to latch the address when it is valid. The 80C188 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80C188 generates two control signals for external transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80C188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80C188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. External circuitry must arbitrate which external device will gain control of the bus when there is more than one alternate local bus master. When the 80C188 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD7, A8–A19, S7/$\overline{RFSH}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80C188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is second only to DRAM refresh requests in priority of activity requests the processor may receive. Any bus cycle in progress will be completed before the 80C188 relinquishes the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clock cycles or more if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

If the 80C188 has relinquished the bus and a refresh request is pending, HLDA is removed (driven LOW) to signal the remote processor that the 80C188 wishes to regain control of the bus. The 80C188 will wait until HOLD is removed before taking control of the bus to run the refresh cycle.

Local Bus Controller and Reset

During RESET, the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float them.

NOTE:
$\overline{RD}$, $\overline{UCS}$, $\overline{LCS}$, and $\overline{TEST}$ are provided with internal pullup devices (750Ω ±20%) which are active while $\overline{RES}$ is driven active. These devices prevent the 80C188 from entering any undesired mode of operation if the inputs are left unconnected.

- Drive $\overline{S0}$–$\overline{S2}$ to the inactive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–AD7, A8–A19, S7/$\overline{RFSH}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80C188 integrated peripherals are controlled by 16-bit registers contained within an internal 256-byte control block. The control block may be mapped into either memory or I/O space. Internal logic will recognize control block addresses and respond to bus cycles. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80C188 CPU at any time.

The control block base address is programmed by a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space. If the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode. At RESET, the relocation register is set to 20FFH, which maps the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

CHIP-SELECT/READY GENERATION LOGIC

The 80C188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80C188 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

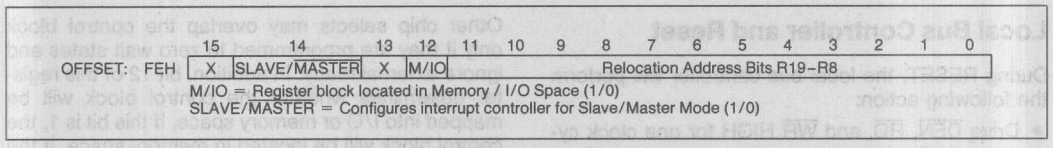


Figure 9. Relocation Register

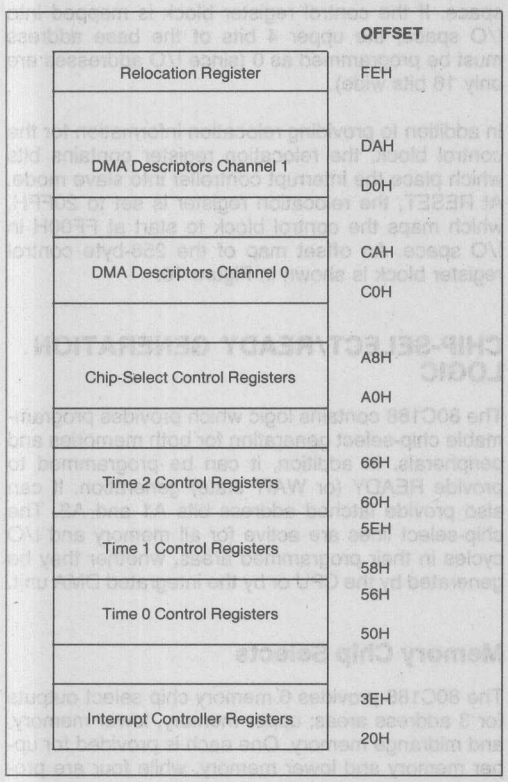


Figure 10. Internal Register Map

Upper Memory $\overline{CS}$

The 80C188 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80C188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

The internal generation of any 20-bit address whose upper 16 bits are equal to or greater than the UMCS value (with bits 0–5 as "0") asserts $\overline{UCS}$. UMCS bits R2–R0 specify the ready mode for the area of memory defined by the chip select register, as explained later.

Lower Memory $\overline{CS}$

The 80C188 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12) at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will assert $\overline{LCS}$. LMCS register bits R2–R0 specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80C188 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located within the 80C188 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. If the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

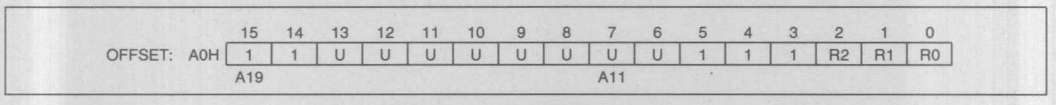


Figure 11. UMCS Register

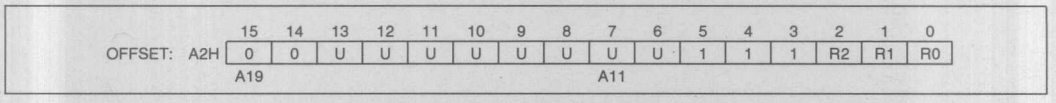


Figure 12. LMCS Register

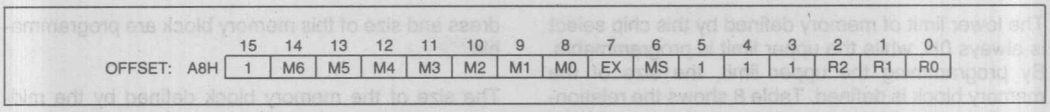


Figure 13. MPCS Register

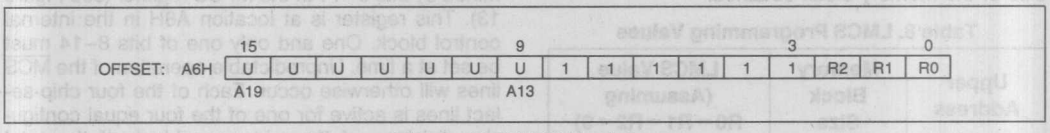


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all four mid-range chip selects.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The 80C188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. The base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80C188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1 and A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0 and A1 pins used for selecting internal registers of 8-bit peripheral chips.

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). The register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

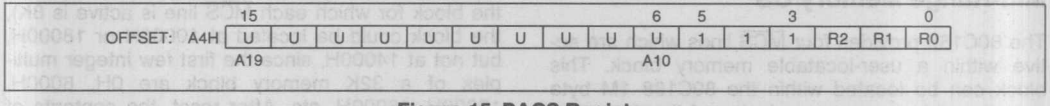


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

$\overline{PCS}$ Line	Active between Locations
$\overline{PCS0}$	PBA —PBA + 127
$\overline{PCS1}$	PBA + 128—PBA + 255
$\overline{PCS2}$	PBA + 256—PBA + 383
$\overline{PCS3}$	PBA + 384—PBA + 511
$\overline{PCS4}$	PBA + 512—PBA + 639
$\overline{PCS5}$	PBA + 640—PBA + 767
$\overline{PCS6}$	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). The register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 specify the READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80C188 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80C188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80C188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series, if the external READY is used (R2 = 0). For example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon RESET, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to insert 3 wait states in conjunction with external READY (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

The 80C188 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data.

DMA Operation

Each channel has six registers in the control block which define each channel's operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Count Register, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

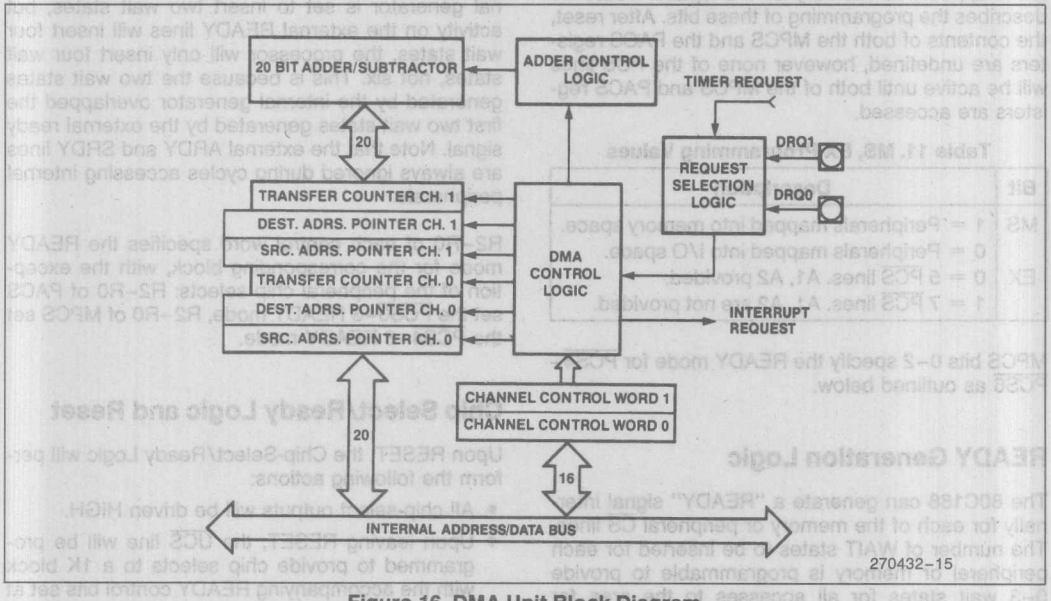


Figure 16. DMA Unit Block Diagram

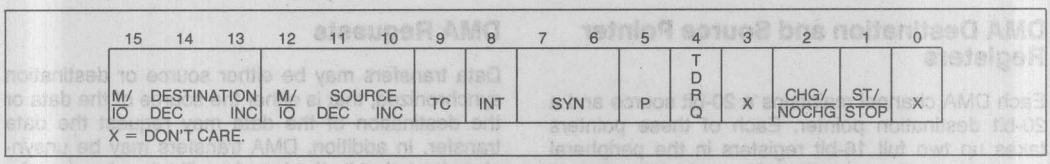


Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80C188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

DEST: M/$\overline{IO}$ Destination pointer is in memory (1) or I/O (0) space.

DEC Decrement destination pointer by 1 after each transfer.

INC Increment destination pointer by 1 after each transfer.

If both INC and DEC are specified, the pointer will not be changed after each cycle.

SOURCE: M/$\overline{IO}$ Source pointer is in memory (1) or I/O (0) space.

DEC Decrement source pointer by 1 after each transfer.

INC Increment source pointer by 1 after each transfer.

If both INC and DEC are specified, the pointer will not be changed after each cycle.

TC: If set, DMA will terminate when the contents of the transfer count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point. If cleared, the DMA controller will decrement the transfer count register for each DMA cycle, but the DMA transfers will not stop when the transfer count register reaches zero.

INT: Enable interrupts to CPU upon transfer count termination.

SYN: 00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST/$\overline{STOP}$ bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

P: Channel priority relative to other channel.

0 Low priority.

1 High priority.

Channels will alternate cycles if both are set at the same priority level.

TDRQ: Enable/Disable (1/0) DMA requests from timer 2.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) the ST/$\overline{STOP}$ bit. If this bit is set when writing the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be read as 0.

ST/$\overline{STOP}$: Start/Stop (1/0) channel.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. For each DMA Channel to be used, all four pointer registers must be initialized. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer.

Each pointer may point into either memory or I/O space. Since the upper four bits of the address are not automatically programmed to zero, the user must program them in order to address the normal 64K I/O space. There is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). The register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). When destination synchronization is performed, data will not be fetched from the source address until the destination device signals that it is ready to receive it. Also, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another destination synchronized DMA cycle will begin after two processor clocks. This allows the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. Table 14 shows the maximum DMA transfer rates.

Table 14. Maximum DMA Transfer Rates at 16 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2.0 MBytes/sec	2.0 MBytes/sec
Source Synch	2.0 MBytes/sec	2.0 MBytes/sec
Destination Synch	1.3 MBytes/sec	1.6 MBytes/sec

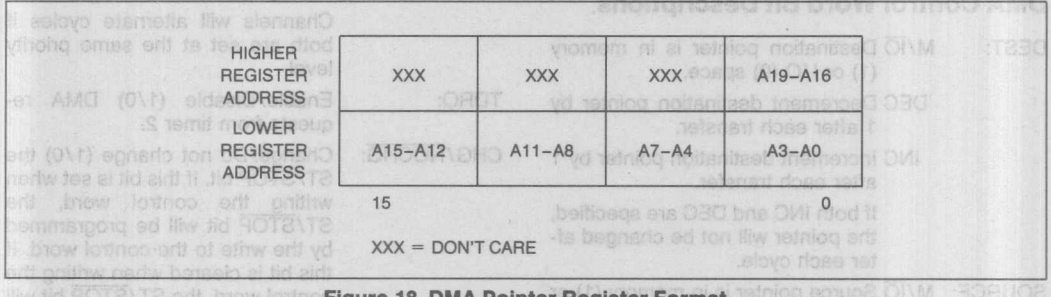


Figure 18. DMA Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed to give one channel priority over the other, or they may be programmed to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses; also, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/$\overline{\text{STOP}}$ bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also be generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before the ST/STOP bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the state of the DMA channels will be as follows:

- The ST/$\overline{\text{STOP}}$ bit for each channel will be reset to $\overline{\text{STOP}}$.
- Any transfer in progress is aborted.
- The values of the transfer count registers, source pointers and destination pointers are indeterminate.

TIMERS

The 80C188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, the third timer can be used as a prescaler to the other two, or as a DMA request source.

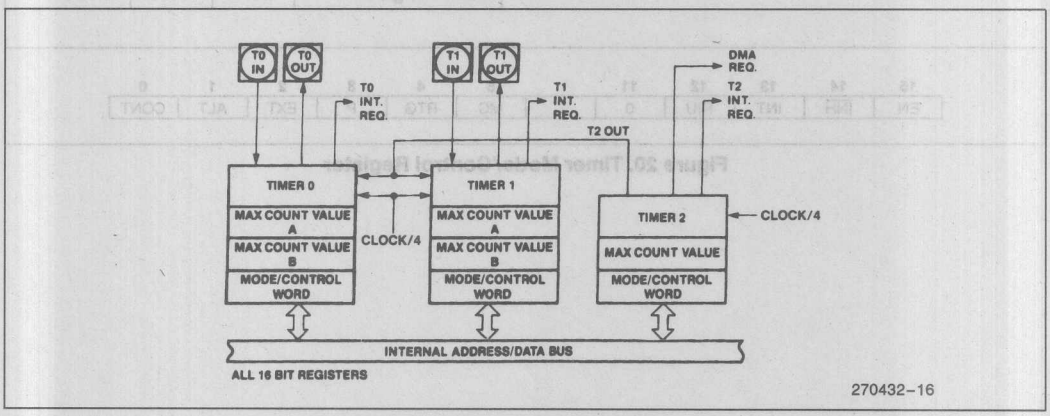


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate. Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

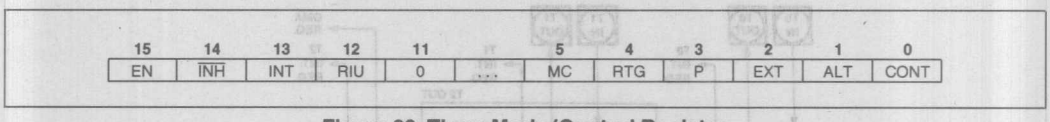


Figure 20. Timer Mode/Control Register

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{\text{INH}}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{\text{INH}}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX CONT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80C188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80C188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The contents of this register may be read or written by the processor at any time. If the register is written while the timer is counting,the new value will take effect in the current count cycle.

The count registers should be programmed before attempting to use the timers since they are not automatically initialized to zero.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. Upon RESET, the state of the timers will be as follows:

- All EN (Enable) bits are reset preventing timer counting.
- For Timers 0 and 1, the RIU bits are reset to zero and the ALT bits are set to one. This results in the Timer Out pins going HIGH.
- The contents of the count registers are indeterminate.

Timers and Reset

A timer resets when the timer count register equals the max count value being used. If the timer count register or the max count register is changed so that the max count is less than the timer count, the timer does not immediately reset. Instead, the timer counts up to 0FFFFH, "wraps around" to zero, counts up to the max count value, and then resets.

INTERRUPT CONTROLLER

The 80C188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80C188 interrupt controller has its own control register that sets the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80C188 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

Five pins are provided for external interrupt sources. One of these pins is NMI, the non-maskable interrupt. NMI is generally used for unusual events such as power-fail interrupts. The other four pins may be configured in any of the following ways:

- As four interrupt input lines with internally generated interrupt vectors.
- As an interrupt line and interrupt acknowledge line pair (cascade mode) with externally generated interrupt vectors plus two interrupt input lines with internally generated vectors.
- As two pairs of interrupt/interrupt acknowledge lines (Cascade Mode) with externally generated interrupt vectors.

External sources in the Cascade Mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80C188 on the second cycle. The capability to interface to external 82C59A programmable interrupt controllers is provided when the inputs are configured in Cascade Mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 82C59A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled, yet be suspended only by interrupts of higher priority than the in-service interrupt.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is executed at the end of the service routine just before the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80C188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the Cascade Mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 82C59A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the Cascade or non-Cascade Mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary Cascade Mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 82C59As. Three levels of priority are created, requiring priority resolution in the 80C188 interrupt controller, the master 82C59As, and the slave 82C59As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

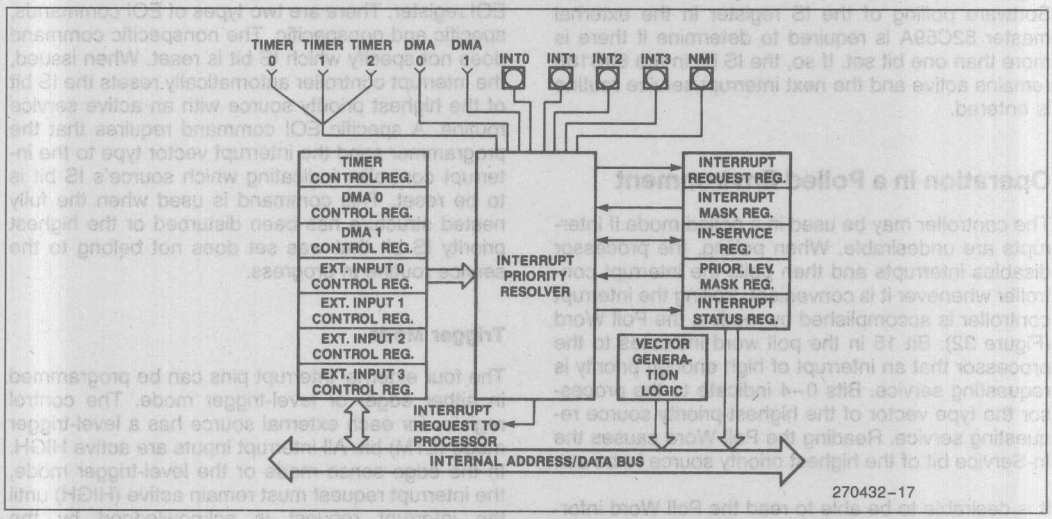


Figure 21. Interrupt Controller Block Diagram

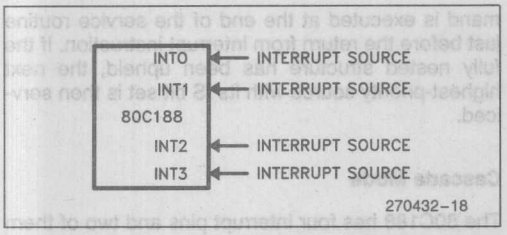


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 82C59A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80C188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80C188 controller until the 80C188 in-service bit is reset. In special fully nested mode, the 80C188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80C188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the IS register in the external master 82C59A is required to determine if there is more than one bit set. If so, the IS bit in the 80C188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80C188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, other interrupt requests can be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the

80C188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80C186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits are set when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corre-

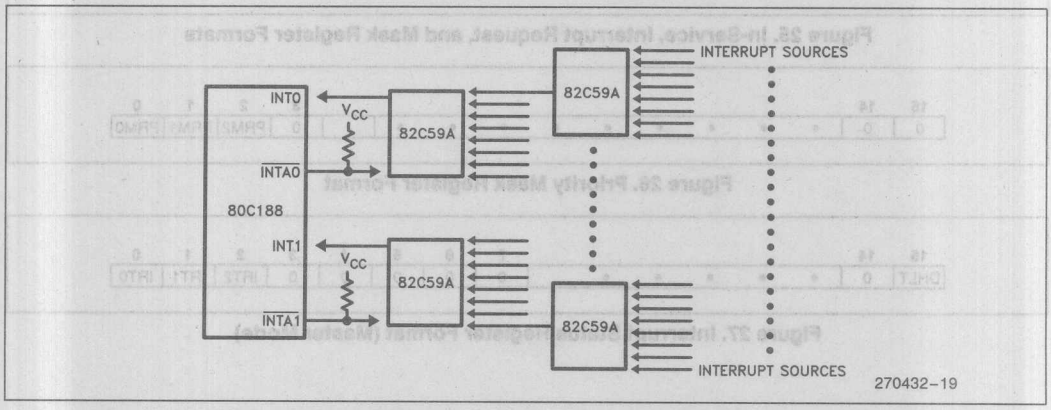


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

sponding to a particular source masks the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

15	14	•	•	•	10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14	•	•	•	•	•	•	•	•	•		3	2	1	0
0	0	•	•	•	•	•	•	•	•	•		0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14	•	•	•	•	•	7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Priority Mask Register

This register masks all interrupts below a particular interrupt priority level. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. This bit allows prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits differentiate between timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80C188 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

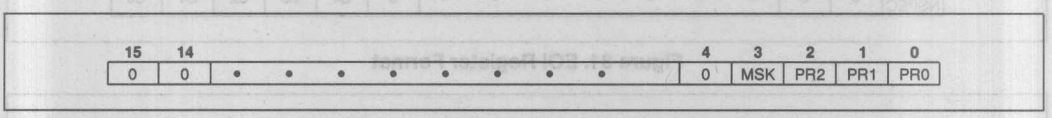


Figure 28. Timer/DMA Control Registers Formats

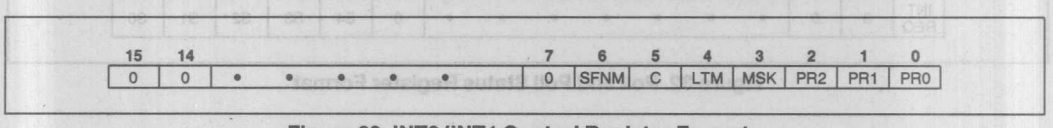


Figure 29. INT0/INT1 Control Register Formats

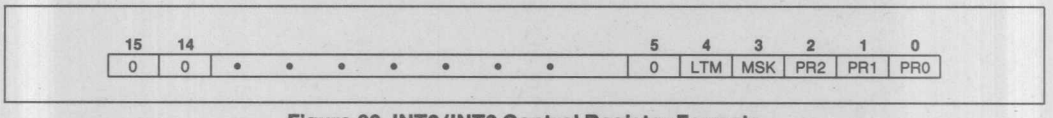


Figure 30. INT2/INT3 Control Register Formats

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_x: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80C188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80C188 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80C188 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 82C59A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80C188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

Slave Mode External Interface

The configuration of the 80C188 with respect to an external 82C59A master is shown in Figure 33. The INT0 (Pin 45) input is used as the 80C188 CPU interrupt input. INT3 (Pin 41) functions as an output to send the 80C188 slave-interrupt-request to one of the 8 master-PIC-inputs.

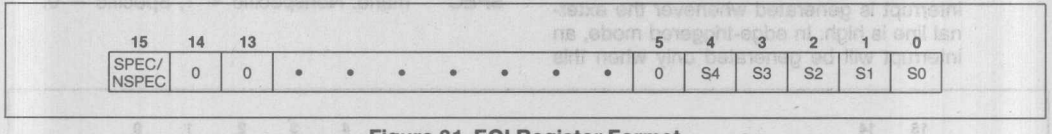


Figure 31. EOI Register Format

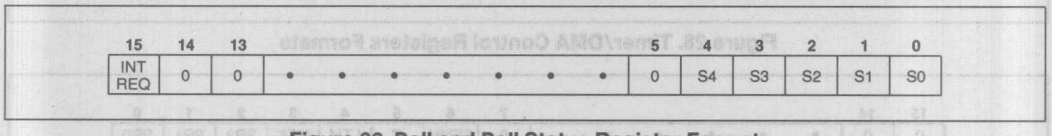


Figure 32. Poll and Poll Status Register Format

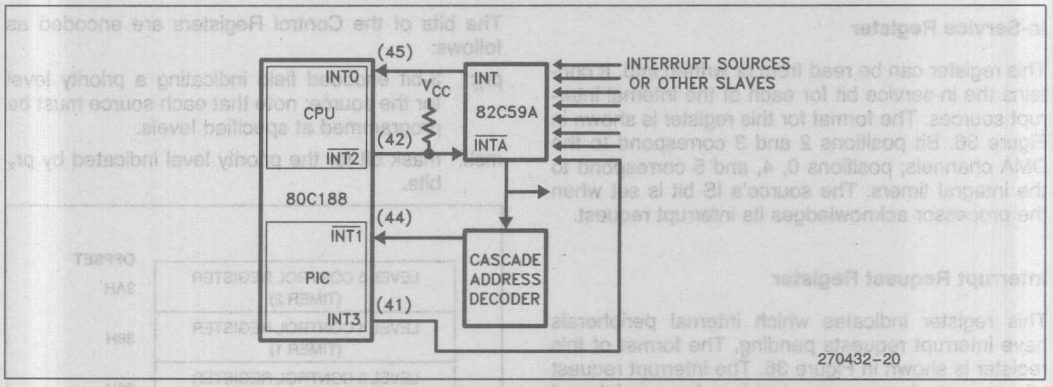


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 82C59As do this internally. Because of pin limitations, the 80C188 slave address will have to be decoded externally. $\overline{INT1}$ (Pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ (Pin 42) is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 82C59A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 82C59A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80C188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented (see Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

Enhanced Mode Operation

In Compatible Mode the 80C188 operates with all the features of the NMOS 80188, with the exception of 8087 support (i.e. no numeric coprocessing is possible). Queue-Status information is still available for design purposes other than 8087 support.

All the Enhanced Mode features are completely masked when in Compatible Mode. A write to any of the Enhanced Mode registers will have no effect, while a read will not return any valid data.

In Enhanced Mode, the 80C188 will operate with Power-Save and DRAM refresh, in addition to all the Compatible Mode features.

Entering Enhanced Mode

Enhanced mode can be entered by tying the RESET output signal from the 80C188 to the TEST/BUSY input.

Queue-Status Mode

The queue-status mode is entered by strapping the $\overline{RD}$ pin low. $\overline{RD}$ is sampled at RESET and if LOW, the 80C188 will reconfigure the ALE and $\overline{WR}$ pins to be QS0 and QS1 respectively. This mode is available on the 80C188 in both Compatible and Enhanced Modes.

DRAM Refresh Control Unit Description

The Refresh Control Unit (RCU) automatically generates DRAM refresh bus cycles. The RCU operates only in Enhanced Mode. After a programmable period of time, the RCU generates a memory read request to the BIU. If the address generated during a refresh bus cycle is within the range of a properly programmed chip select, that chip select will be activated when the BIU executes the refresh bus cycle. The ready logic and wait states programmed for that region will also be in force. If no chip select is activated, then external ready is automatically required to terminate the refresh bus cycle.

If the HLDA pin is active when a DRAM refresh request is generated (indicating a bus hold condition), then the 80C188 will deactivate the HLDA pin in order to perform a refresh cycle. The circuit external to the 80C188 must remove the HOLD signal in order to execute the refresh cycle. The sequence of HLDA going inactive while HOLD is being held active can be used to signal a pending refresh request.

All registers controlling DRAM refresh may be read and written in Enhanced Mode. When the processor is operating in Compatible Mode, they are deselected and are therefore inaccessible. Some fields of these registers cannot be written and are always read as zeros.

DRAM Refresh Addresses

The address generated during a refresh cycle is determined by the contents of the MDRAM register (see Figure 40) and the contents of a 9-bit counter. Figure 41 illustrates the origin of each bit.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MDRAM: Offset E0H	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0

Bits 0–8: Reserved, read back as 0.

Bits 9–15: M0–M6, are address bits A13–A19 of the 20-bit memory refresh address. These bits should correspond to the chip select address to be activated for the DRAM partition. These bits are set to 0 on RESET.

Figure 40. Memory Partition Register

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
M6	M5	M4	M3	M2	M1	M0	0	0	0	CA8	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0	1

M6–M0: Bits defined by MDRAM Register

CA8–CA0: Bits defined by refresh address counter. These bits do not directly follow a binary count. Rather, they are implemented using a Linear/Feedback Shift Register.

Figure 41. Addresses Generated by RCU

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDRAM: Offset E2H	0	0	0	0	0	0	0	C8	C7	C6	C5	C4	C3	C2	C1	C0

Bits 0–8: C0–C8, clock divisor register, holds the number of CLKOUT cycles between each refresh request.

Bits 9–15: Reserved, read back as 0.

Figure 42. Clock Pre-Scaler Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EDRAM: Offset E4H	E	0	0	0	0	0	0	T8	T7	T6	T5	T4	T3	T2	T1	T0

Bits 0–8: T0–T8, refresh clock counter outputs. Read only.

Bits 9–14: Reserved, read back as 0.

Bit 15: Enable RCU, set to 0 on RESET.

Figure 43. Enable RCU Register

Refresh Control Unit Programming and Operation

After programming the MDRAM and the CDRAM registers (Figures 40 and 42), the RCU is enabled by setting the "E" bit in the EDRAM register (Figure 43). The clock counter (T0–T8 of EDRAM) will be loaded from C0–C8 of CDRAM during T_3 of instruction cycle that sets the "E" bit. The clock counter is then decremented at each subsequent CLKOUT.

A refresh is requested when the value of the counter has reached 1 and the counter is reloaded from CDRAM. In order to avoid missing refresh requests, the value in the CDRAM register should always be at least 18 (12H). Clearing the "E" bit at anytime will clear the counter and stop refresh requests, but will not reset the refresh address counter.

POWER-SAVE CONTROL

Power Save Operation

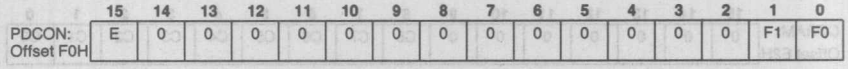
The 80C188, when in Enhanced Mode, can enter a power saving state by internally dividing the clock-in frequency by a programmable factor. This divided frequency is also available at the CLKOUT pin. The PDCON register contains the two-bit fields for selecting the clock division factor and the enable bit.

All internal logic, including the Refresh Control Unit and the timers, will have their clocks slowed down by the division factor. To maintain a real time count or a fixed DRAM refresh rate, these peripherals must be re-programmed when entering and leaving the power-save mode.

The power-save mode is exited whenever an interrupt is processed by automatically resetting the enable bit. If the power-save mode is to be re-entered after serving the interrupt, the enable bit will need to be set in software before returning from the interrupt routine.

The internal clocks of the 80C188 will begin to be divided during the T_3 state of the instruction cycle that sets the enable bit. Clearing the enable bit will restore full speed in the T_3 state of that instruction.

At no time should the internal clock frequency be allowed to fall below 0.5 MHz. This is the minimum operational frequency of the 80C188. For example, an 80C188 running with a 12 MHz crystal (6 MHz CLOCKOUT) should never have a clock divisor greater than eight.



Bits 0–1: Clock Divisor Select

F1	F0	Division Factor
0	0	divide by 1
0	1	divide by 4
1	0	divide by 8
1	1	divide by 16

Bits 2–14: Reserved, read back as zero.

Bit 15: Enable Power Save Mode. Set to zero on RESET.

Figure 44. Power-Save Control Register

ONCE™ Test Mode

To facilitate testing and inspection of devices when fixed into a target system, the 80C188 has a test mode available which allows all pins to be placed in a high-impedance state. ONCE stands for "ON Circuit Emulation". When placed in this mode, the 80C188 will put all pins in the high-impedance state until RESET.

The ONCE mode is selected by tying the $\overline{UCS}$ and the $\overline{LCS}$ LOW during RESET. These pins are sampled on the low-to-high transition of the $\overline{RES}$ pin. The $\overline{UCS}$ and the $\overline{LCS}$ pins have weak internal pull-up resistors similar to the $\overline{RD}$ and $\overline{TEST}$ pins to guarantee normal operation.

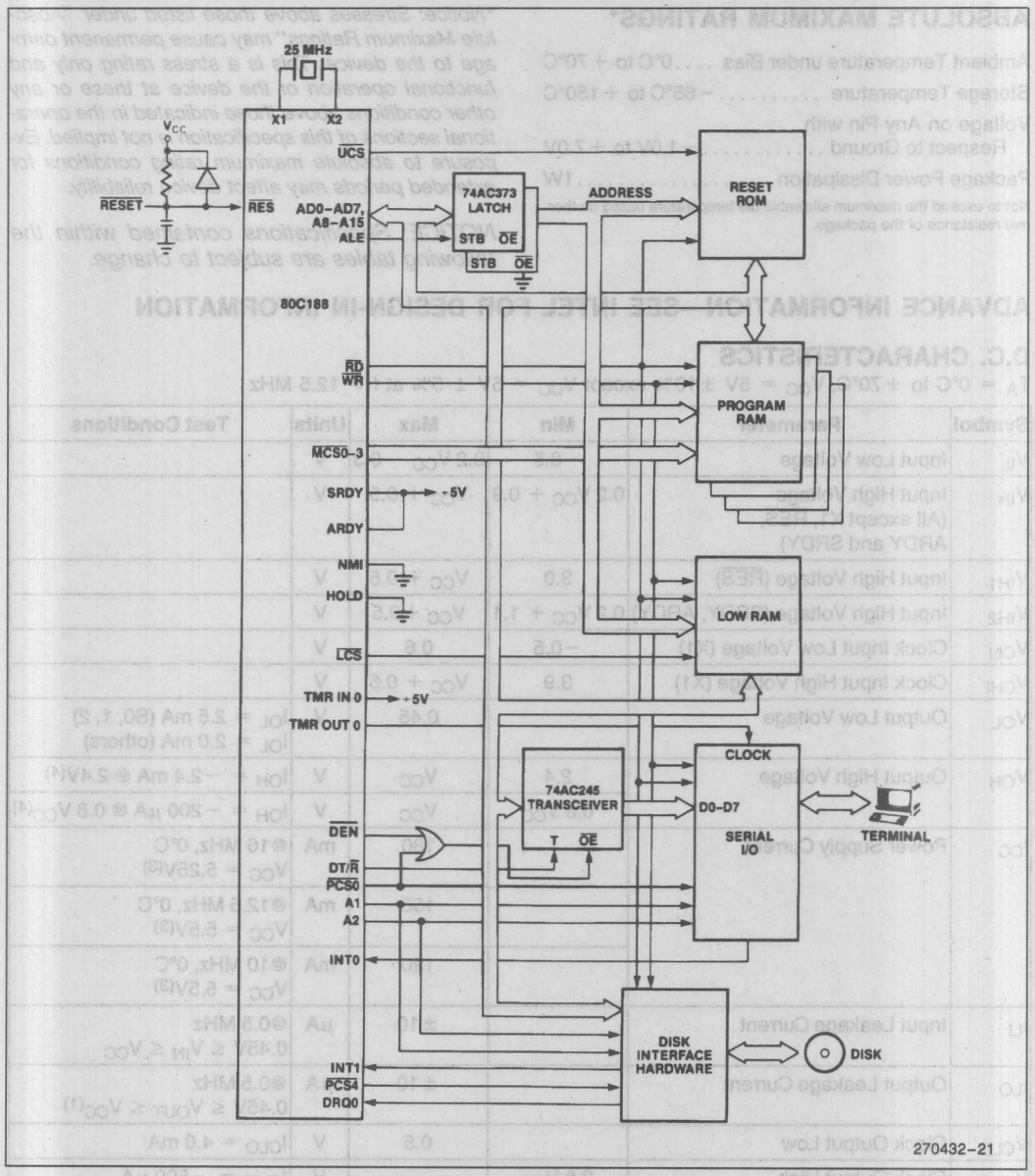


Figure 45. Typical 80C188 Computer

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .... 0°C to +70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with Respect to Ground ..... −1.0V to +7.0V
Package Power Dissipation 1W
Not to exceed the maximum allowable die temperature based on thermal resistance of the package.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ± 5% at f > 12.5 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.2 V_{CC} − 0.3	V	
V_{IH}	Input High Voltage (All except X1, $\overline{RES}$, ARDY and SRDY)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{IH2}	Input High Voltage (SRDY, ARDY)	0.2 V_{CC} + 1.1	V_{CC} +0.5	V	
V_{CLI}	Clock Input Low Voltage (X1)	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage (X1)	3.9	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA (S0, 1, 2) I_{OL} = 2.0 mA (others)
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −2.4 mA @ 2.4V(4)
		0.8 V_{CC}	V_{CC}	V	I_{OH} = −200 μA @ 0.8 V_{CC}(4)
I_{CC}	Power Supply Current		180	mA	@16 MHz, 0°C V_{CC} = 5.25V(3)
			150	mA	@12.5 MHz, 0°C V_{CC} = 5.5V(3)
			120	mA	@10 MHz, 0°C V_{CC} = 5.5V(3)
I_{LI}	Input Leakage Current		±10	μA	@0.5 MHz 0.45V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	@0.5 MHz 0.45V ≤ V_{OUT} ≤ V_{CC}(1)
V_{CLO}	Clock Output Low		0.5	V	I_{CLO} = 4.0 mA
V_{CHO}	Clock Output High	0.8 V_{CC}		V	I_{CHO} = −500 μA
C_{IN}	Input Capacitance		10	pF	@ 1 MHz(2)
C_{IO}	I/O Capacitance		20	pF	@ 1 MHz(2)

NOTES:
1. Pins being floated during HOLD or by invoking the ONCE Mode.
2. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at +5.0V or 0.45V. This parameter is not tested.
3. Current is measured with the device in RESET with X1 and X2 driven and all other non-power pins open.
4. $\overline{RD}$/QSMD, UCS, LCS, $\overline{TEST}$ pins have internal pullup devices that are active at RESET. Excessive loading on these pins can cause the 80C188 to go into undesired modes of operation (e.g. Queue Status, ONCE) upon RESET.

POWER SUPPLY CURRENT

Current is linearly proportional to clock frequency and is measured with the device in RESET with X1 and X2 driven and all other non-power pins open.

Maximum current is indicated in the D.C. Characteristics.

Typical current is given by I_{CC} (typ.) = 6.4 mA X freq. (MHz) +4.0 mA. "Typicals" are based on a limited number of samples taken from early manufacturing lots measured at V_{CC} = 5V and room temperature. "Typicals" are not guaranteed.

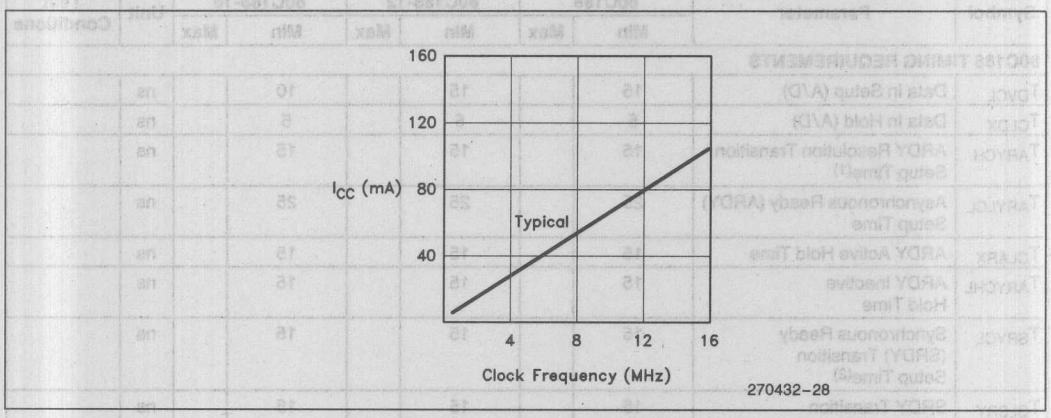


Figure 46. I_{CC} (typical) vs Frequency

PIN TIMINGS

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz
All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} − 0.5V.

Symbol	Parameter	80C188		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 TIMING REQUIREMENTS									
T_{DVCL}	Data In Setup (A/D)	15		15		10		ns	
T_{CLDX}	Data In Hold (A/D)	5		5		5		ns	
T_{ARYCH}	ARDY Resolution Transition Setup Time(1)	15		15		15		ns	
T_{ARYLCL}	Asynchronous Ready (ARDY) Setup Time	25		25		25		ns	
T_{CLARX}	ARDY Active Hold Time	15		15		15		ns	
T_{ARYCHL}	ARDY Inactive Hold Time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time(2)	15		15		15		ns	
T_{CLSRY}	SRDY Transition Hold Time(2)	15		15		15		ns	
T_{HVCL}	HOLD Setup(1)	15		15		15		ns	
T_{INVCH}	INTR, NMI, TEST, TMR IN Setup Time(1)	15		15		15		ns	
T_{INVCL}	DRQ0, DRQ1, $\overline{RES}$, Setup Time(1)	15		15		15		ns	
80C188 MASTER INTERFACE TIMING RESPONSES									
T_{CLAV}	Address Valid Delay	5	50	5	36	5	33	ns	C_L = 50 pF –200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLAX}	Address Hold	0		0		0		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	30	T_{CLAX}	25	T_{CLAX}	20	ns	
T_{CHCZ}	Command Lines Float Delay		40		33		28	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		45		37		32	ns	
T_{LHLL}	ALE Width (min)	T_{CLCL} − 30		T_{CLCL} − 30		T_{CLCL} − 30		ns	C_L = 50 pF –100 pF all outputs @ 12.5 & 16 MHz
T_{CHLH}	ALE Active Delay		30		25		20	ns	
T_{CHLL}	ALE Inactive Delay		30		25		20	ns	
T_{LLAX}	Address Hold to ALE Inactive (min)	T_{CHCL} − 20		T_{CHCL} − 15		T_{CHCL} − 15		ns	
T_{CLDV}	Data Valid Delay	5	40	5	36	5	33	ns	
T_{CLDOX}	Data Hold Time	3		3		3		ns	
T_{WHDX}	Data Hold after $\overline{WR}$ (min)	T_{CLCL} − 34		T_{CLCL} − 20		T_{CLCL} − 20		ns	
T_{CVCTV}	Control Active Delay 1	3	56	3	47	3	31	ns	
T_{CHCTV}	Control Active Delay 2	5	44	5	37	5	31	ns	
T_{CVCTX}	Control Inactive Delay	3	44	3	37	3	31	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	5	56	5	47	5	35	ns	

NOTES:
1. To guarantee recognition at next clock.
2. To guarantee proper operation.

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} −0.5V.

Symbol	Parameter	80C188		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C188 MASTER INTERFACE TIMING RESPONSES (Continued)									
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		0		ns	C_L=50–200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLRL}	$\overline{RD}$ Active Delay	5	44	5	37	5	31	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	5	44	5	37	5	31	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active (min)	T_{CLCL} − 40		T_{CLCL} − 20		T_{CLCL} − 20		ns	C_L=50–100 pF all outputs @ 12.5 & 16 MHz
T_{CLHAV}	HLDA Valid Delay	3	40	3	33	3	25	ns	
T_{RLRH}	$\overline{RD}$ Pulse Width (min)	$2T_{CLCL}$ − 46		$2T_{CLCL}$ − 40		$2T_{CLCL}$ − 30		ns	
T_{WLWH}	$\overline{WR}$ Pulse Width (min)	$2T_{CLCL}$ − 34		$2T_{CLCL}$ − 30		$2T_{CLCL}$ − 25		ns	
T_{AVLL}	Address Valid to ALE Low (min)	T_{CLCH} − 19		T_{CLCH} − 15		T_{CLCH} − 15		ns	Equal Loading
T_{CHSV}	Status Active Delay	5	45	5	35	5	31	ns	
T_{CLSH}	Status Inactive Delay	5	50	5	35	5	30	ns	
T_{CLTMV}	Timer Output Delay		48		40		30	ns	100 pF max @ 10 MHz
T_{CLRO}	Reset Delay		48		40		30	ns	C_L = 50–200 pF All outputs (except T_{CLTMV}) @ 10 MHz
T_{CHQSV}	Queue Status Delay		28		28		25	ns	
T_{CHDX}	Status Hold Time	5		5		5		ns	
T_{AVCH}	Address Valid to Clock High	0		0		0		ns	C_L = 50–100 pF All outputs @ 12.5 & 16 MHz
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	3	45	3	40	3	35	ns	
T_{DXDL}	$\overline{DEN}$ Inactive to DT/$\overline{R}$ Low	0		0		0		ns	Equal Loading
80C188 CHIP-SELECT TIMING RESPONSES									
T_{CLCSV}	Chip-Select Active Delay		45		33		30	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	T_{CLCH} − 10		T_{CLCH} − 10		T_{CLCH} − 10		ns	Equal Loading
T_{CHCSX}	Chip-Select Inactive Delay	5	32	5	28	5	23	ns	

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at f > 12.5 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
For A.C. tests, input V_{IL} = 0.45V and V_{IH} = 2.4V except at X1 where V_{IH} = V_{CC} −0.5V.

Symbol	Parameter	80C188		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C188 CLKIN REQUIREMENTS Measurements taken with following conditions: External clock input to X1 and X2 not connected (float)									
T_{CKIN}	CLKIN Period	50	1000	40	1000	31.25	1000	ns	
T_{CKHL}	CLKIN Fall Time		5		5		5	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		5		5		5	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	20		16		13		ns	1.5V(2)
T_{CHCK}	CLKIN High Time	20		16		13		ns	1.5V(2)
80C188 CLKOUT TIMING 200 pF load maximum for 10 MHz or less, 100 pF load maximum above 10 MHz									
T_{CICO}	CLKIN to CLKOUT Skew		25		21		17	ns	
T_{CLCL}	CLKOUT Period	100	2000	80	2000	62.5	2000	ns	
T_{CLCH}	CLKOUT	$0.5\,T_{CLCL}-8$		$0.5\,T_{CLCL}-7$		$0.5\,T_{CLCL}-7$		ns	C_L = 100 pF(2)
	Low Time (min)	$0.5\,T_{CLCL}-6$		$0.5\,T_{CLCL}-5$		$0.5\,T_{CLCL}-5$		ns	C_L = 50 pF(3)
T_{CHCL}	CLKOUT High Time (min)	$0.5\,T_{CLCL}-8$		$0.5\,T_{CLCL}-7$		$0.5\,T_{CLCL}-7$		ns	C_L = 100 pF(4)
		$0.5\,T_{CLCL}-6$		$0.5\,T_{CLCL}-5$		$0.5\,T_{CLCL}-5$		ns	C_L = 50 pF(3)
T_{CH1CH2}	CLKOUT Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		10		10		8	ns	3.5 to 1.0V

NOTES:
1. T_{CLCK} and T_{CHCK} (CLKIN Low and High times) should not have a duration less than 40% of T_{CKIN}.
2. Tested under worst case conditions: V_{CC} = 5.5V (5.25V @16 MHz), T_A = 70°C.
3. Not Tested.
4. Tested under worst case conditions: V_{CC} = 4.5V (4.75V @16 MHz), T_A = 0°C.

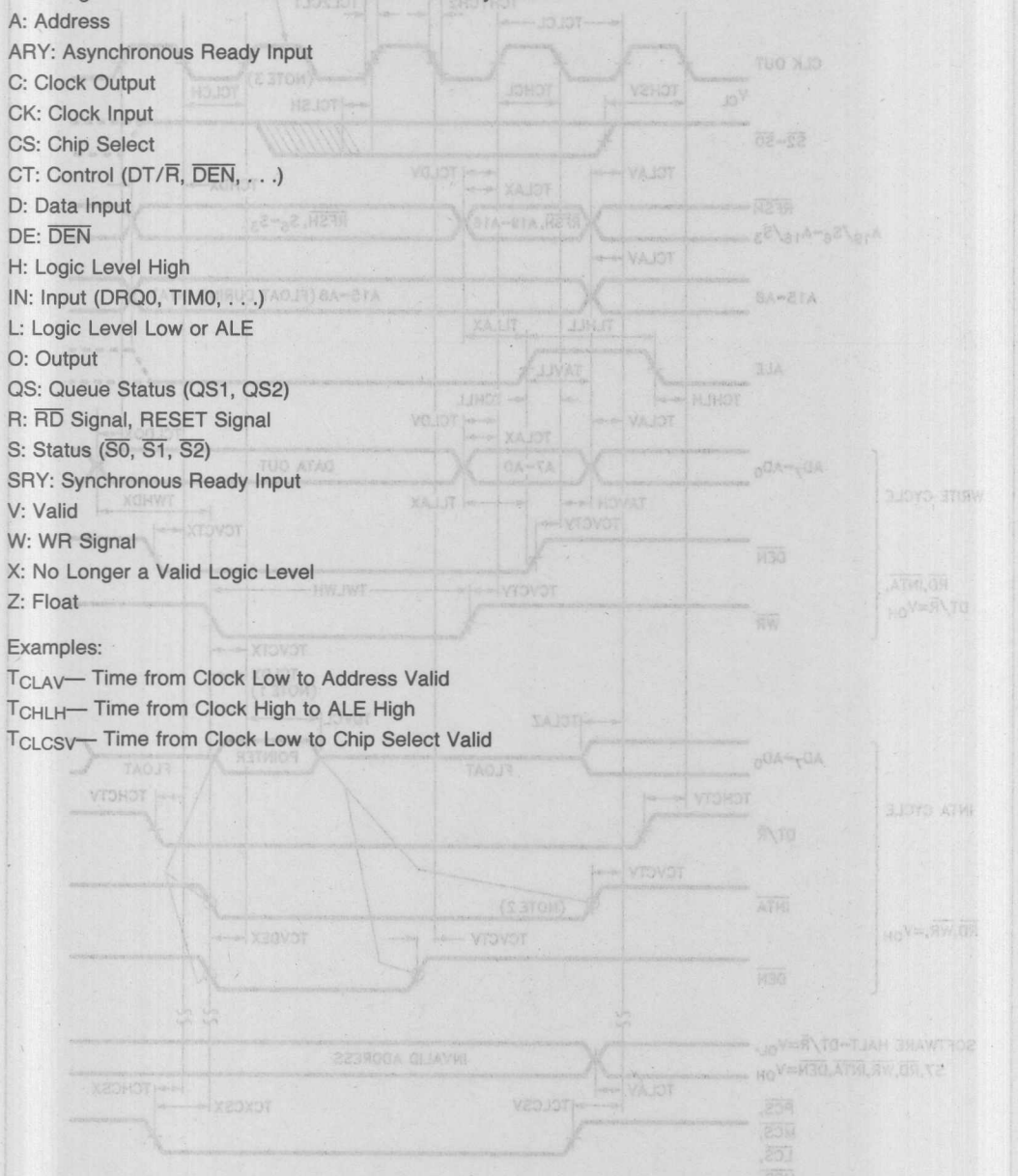
EXPLANATION OF THE AC SYMBOLS

Each timing symbol has from 5 to 7 characters. The first character is always a "T" (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

A: Address
ARY: Asynchronous Ready Input
C: Clock Output
CK: Clock Input
CS: Chip Select
CT: Control (DT/$\overline{R}$, $\overline{DEN}$, . . .)
D: Data Input
DE: $\overline{DEN}$
H: Logic Level High
IN: Input (DRQ0, TIM0, . . .)
L: Logic Level Low or ALE
O: Output
QS: Queue Status (QS1, QS2)
R: $\overline{RD}$ Signal, RESET Signal
S: Status ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$)
SRY: Synchronous Ready Input
V: Valid
W: WR Signal
X: No Longer a Valid Logic Level
Z: Float

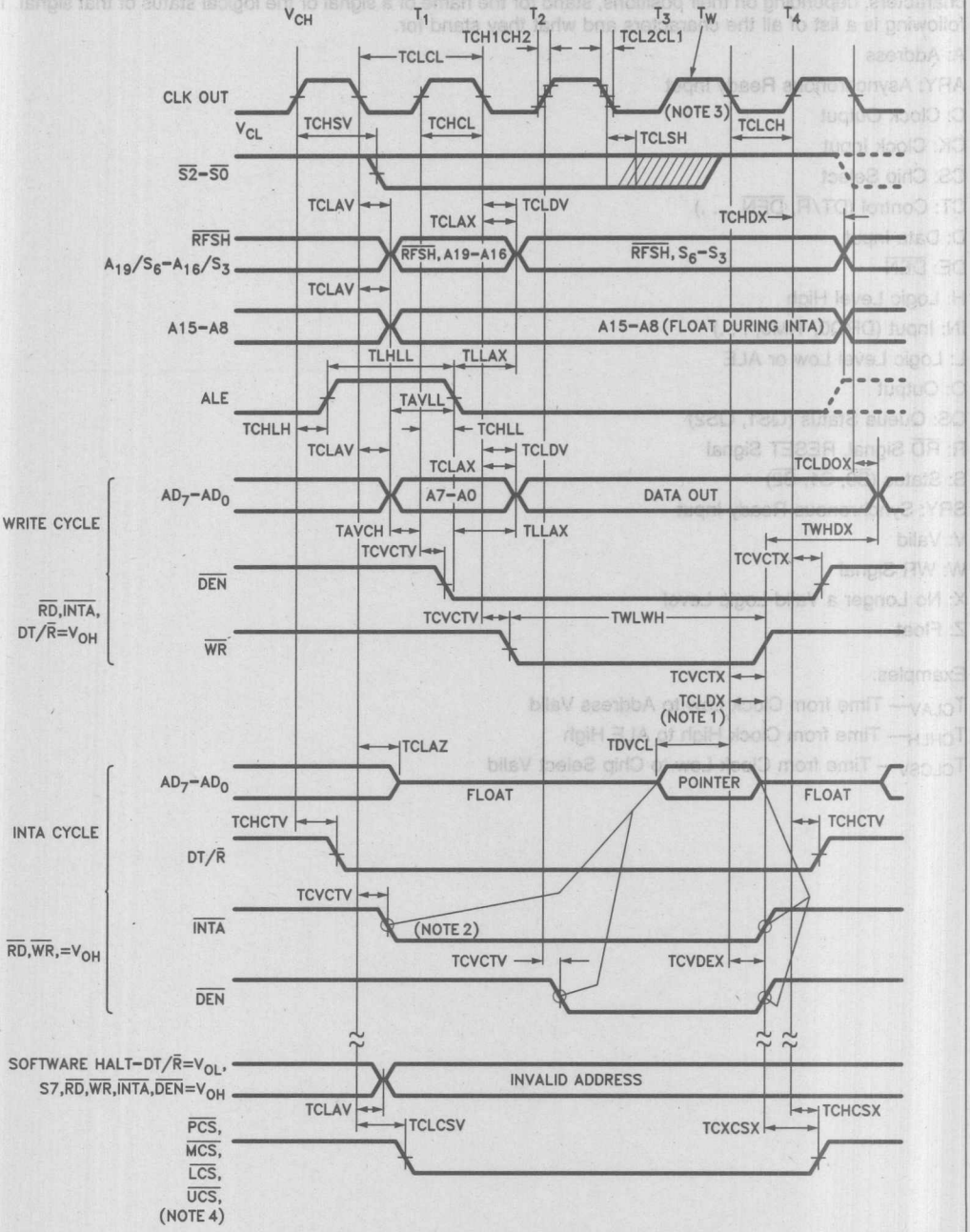
Examples:

T_{CLAV}— Time from Clock Low to Address Valid
T_{CHLH}— Time from Clock High to ALE High
T_{CLCSV}— Time from Clock Low to Chip Select Valid



WAVEFORMS

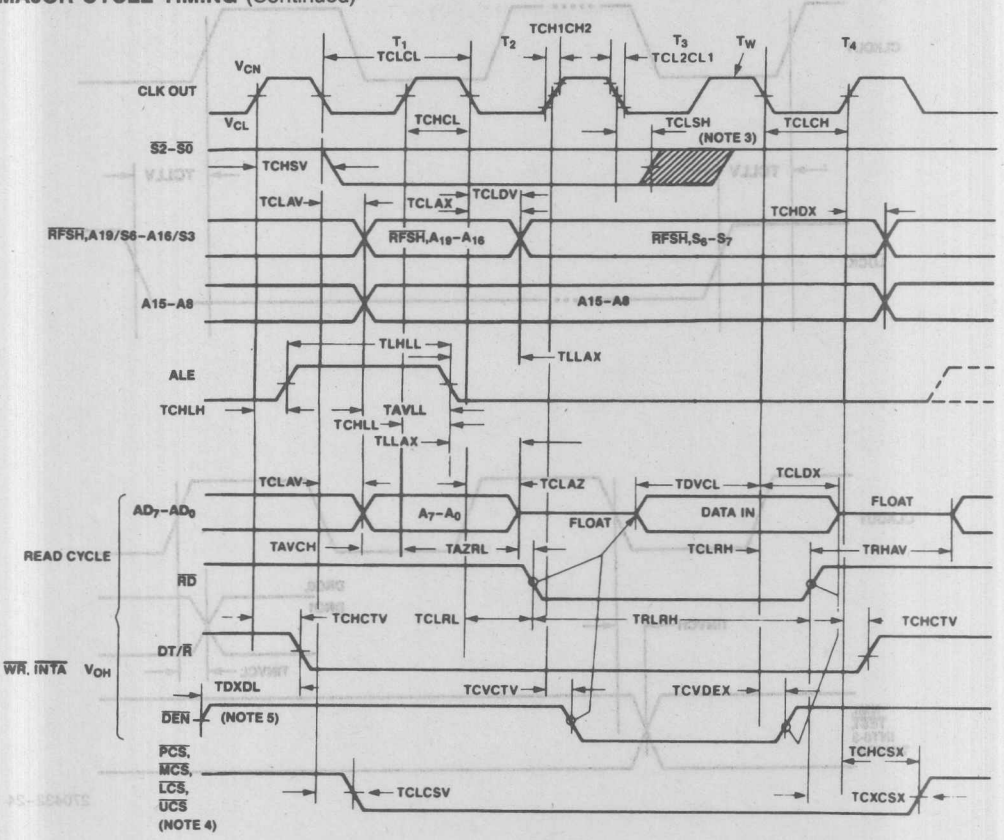
MAJOR CYCLE TIMING



270432-22

WAVEFORMS (Continued)

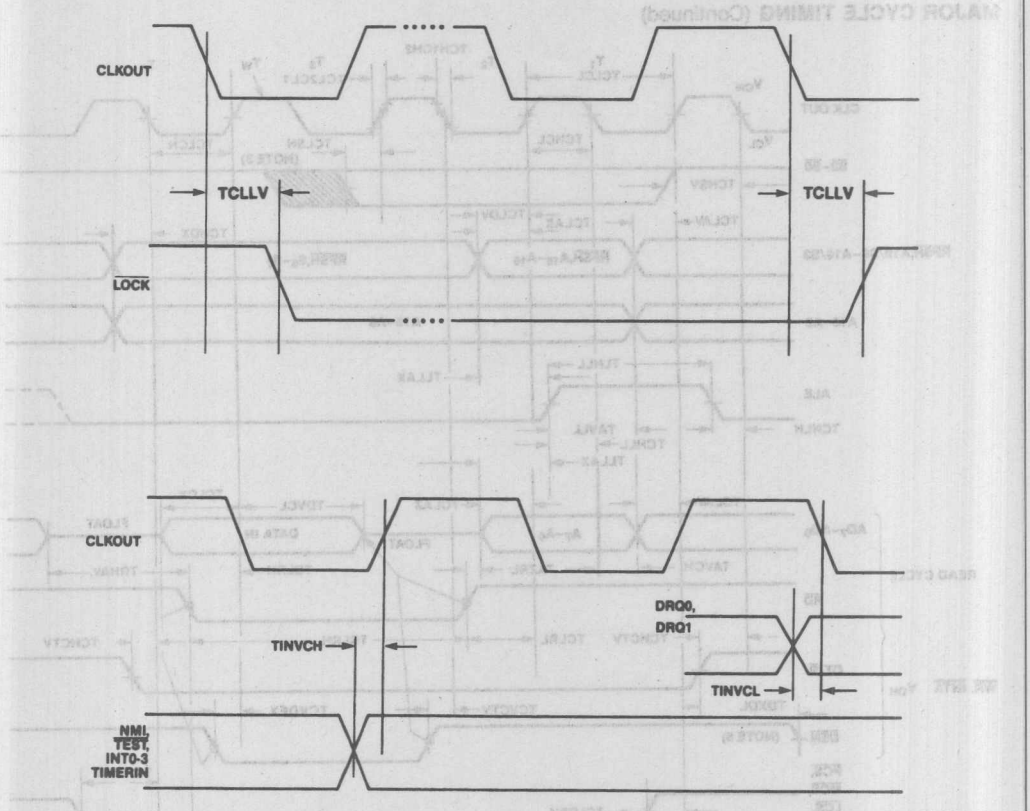
MAJOR CYCLE TIMING (Continued)



270432-23

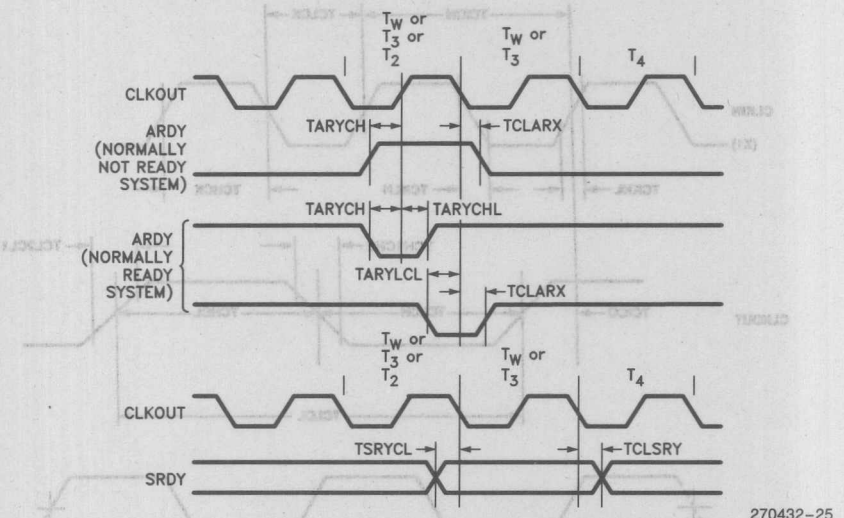
NOTES:
1. The data hold time lasts only until $\overline{INTA}$ goes inactive, even if the $\overline{INTA}$ transition occurs prior to T_{CLDX} (min).
2. INTA occurs one clock later in Slave Mode.
3. Status inactive just prior to T_4.
4. Latched A1 and A2 have the same timings as $\overline{PCS5}$ and $\overline{PCS6}$.
5. For Write cycle followed by Read.

WAVEFORMS (Continued)



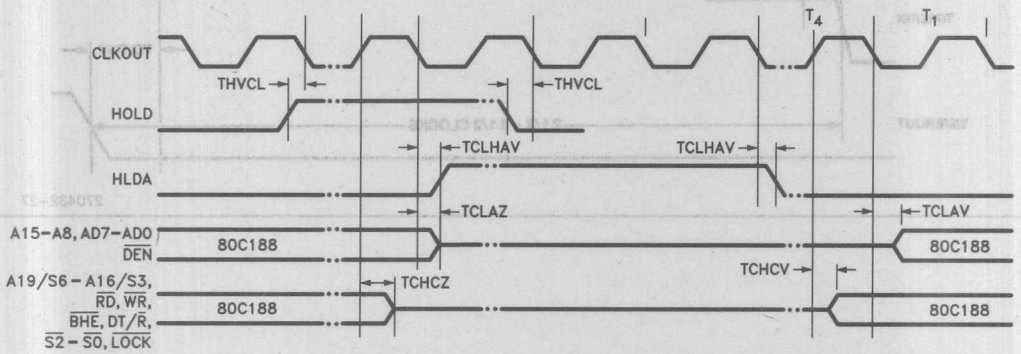
WAVEFORMS (Continued)

READY TIMING



270432-25

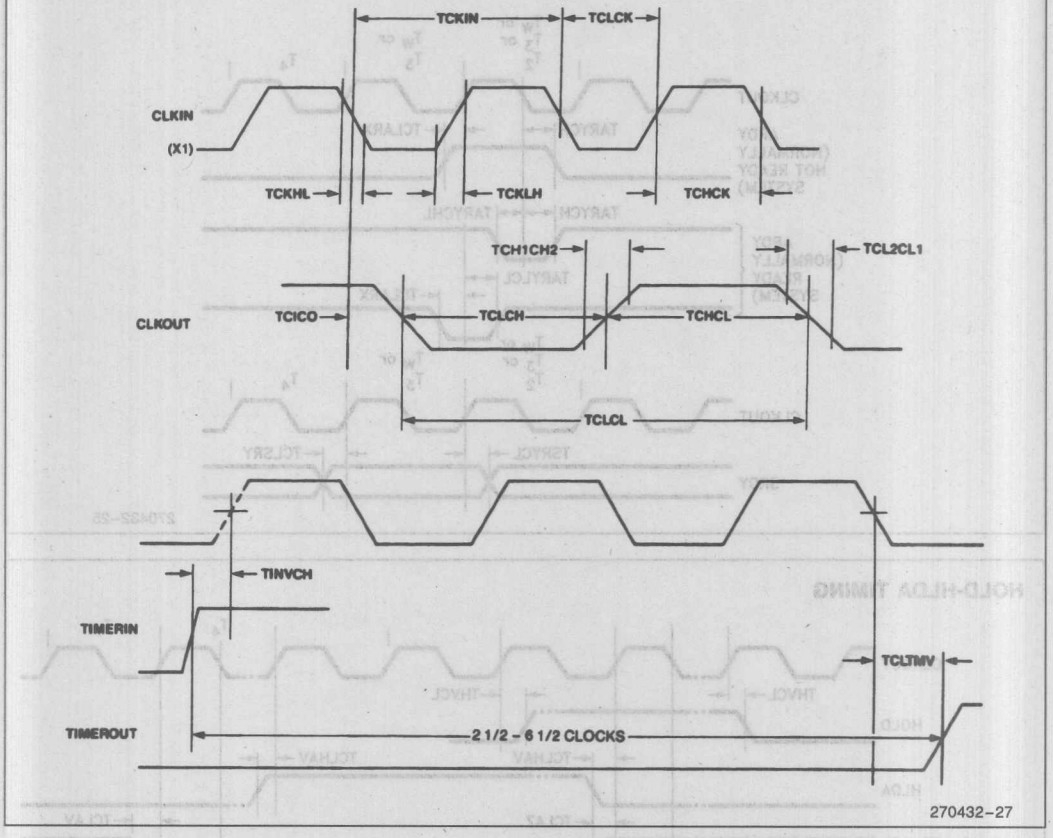
HOLD-HLDA TIMING



270432-26

WAVEFORMS (Continued)

TIMER ON 80C186



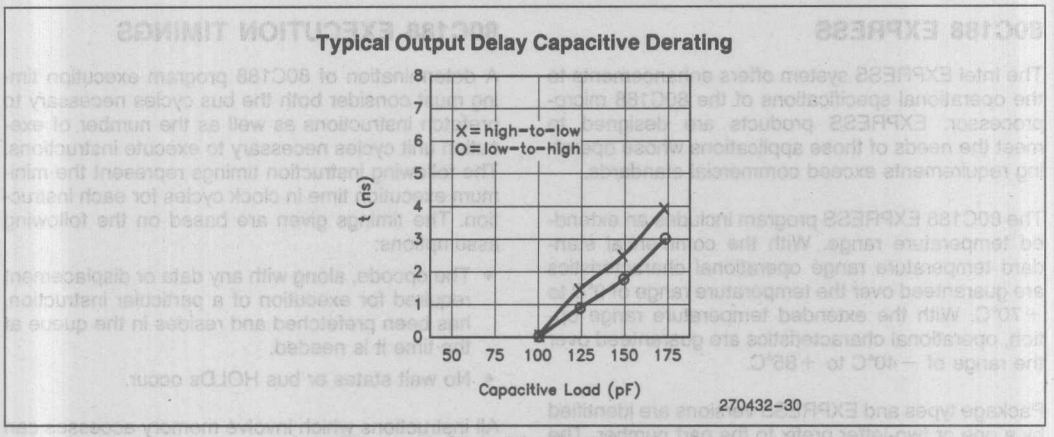


Figure 47. Capacitive Derating Curve

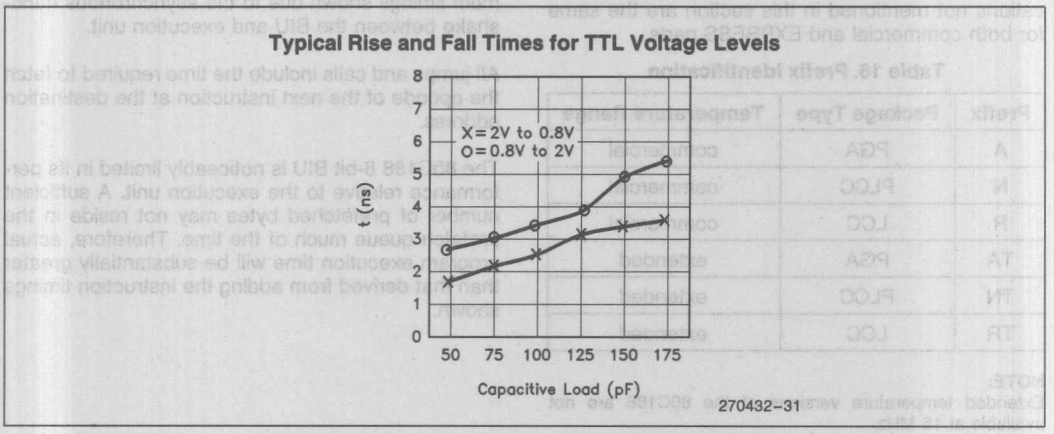


Figure 48. TTL Level Slew Rates for Output Buffers

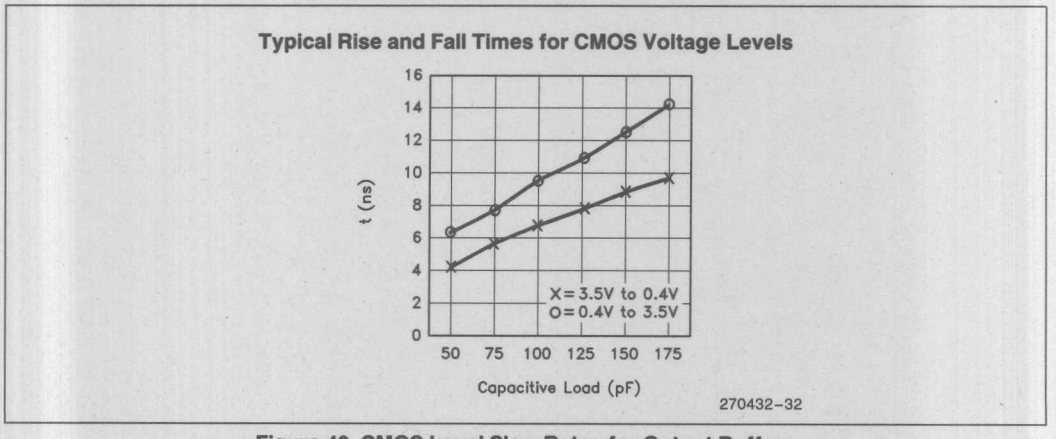


Figure 49. CMOS Level Slew Rates for Output Buffers

80C188 EXPRESS

The Intel EXPRESS system offers enhancements to the operational specifications of the 80C188 microprocessor. EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The 80C188 EXPRESS program includes an extended temperature range. With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to +70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

Package types and EXPRESS versions are identified by a one or two-letter prefix to the part number. The prefixes are listed in Table 16. All AC and DC specifications not mentioned in this section are the same for both commercial and EXPRESS parts.

Table 16. Prefix Identification

Prefix	Package Type	Temperature Range
A	PGA	commercial
N	PLCC	commercial
R	LCC	commercial
TA	PGA	extended
TN	PLCC	extended
TR	LCC	extended

NOTE:
Extended temperature versions of the 80C188 are not available at 16 MHz.

80C188 EXECUTION TIMINGS

A determination of 80C188 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.

All instructions which involve memory accesses can require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

The 80C188 8-bit BIU is noticeably limited in its performance relative to the execution unit. A sufficient number of prefetched bytes may not reside in the prefetch queue much of the time. Therefore, actual program execution time will be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1000100w	mod reg r/m			2/12*	
Register/memory to register	1000101w	mod reg r/m			2/9*	
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1	12/13	8/16-bit
Immediate to register	1011w reg	data	data if w=1		3/4	8/16-bit
Memory to accumulator	1010000w	addr-low	addr-high		8*	
Accumulator to memory	1010001w	addr-low	addr-high		9*	
Register/memory to segment register	10001110	mod 0 reg r/m			2/13	
Segment register to register/memory	10001100	mod 0 reg r/m			2/15	
PUSH = Push:						
Memory	11111111	mod 1 1 0 r/m			20	
Register	01010 reg				14	
Segment register	000 reg 1 1 0				13	
Immediate	011010s0	data	data if s=0		14	
PUSHA = Push All	01100000				68	
POP = Pop:						
Memory	10001111	mod 0 0 0 r/m			24	
Register	01011 reg				14	
Segment register	000 reg 1 1 1	(reg≠01)			12	
POPA = Pop All	01100001				83	
XCHG = Exchange:						
Register/memory with register	1000011w	mod reg r/m			4/17*	
Register with accumulator	10010 reg				3	
IN = Input from:						
Fixed port	1110010w	port			10*	
Variable port	1110110w				8*	
OUT = Output to:						
Fixed port	1110011w	port			9*	
Variable port	1110111w				7*	
XLAT = Translate byte to AL	11010111				15	
LEA = Load EA to register	10001101	mod reg r/m			6	
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	10011111				2	
SAHF = Store AH into flags	10011110				3	
PUSHF = Push flags	10011100				13	
POPF = Pop flags	10011101				12	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 000 r/m			3/15*	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1111111w	mod 001 r/m			3/15*	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10*	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43*	
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43*	
IMUL = Integer Immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44*	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67*	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
	TTT Instruction					
	0 0 0 ROL					
	0 0 1 ROR					
	0 1 0 RCL					
	0 1 1 RCR					
	1 0 0 SHL/SAL					
	1 0 1 SHR					
	1 1 1 SAR					
AND = **And:**						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
TEST = And function to flags, no result:						
Register/memory and register	1000010w	mod reg r/m			3/10*	
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w=1	4/10*	
Immediate data and accumulator	1010100w	data	data if w=1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	000010dw	mod reg r/m			3/10*	
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0000110w	data	data if w=1		3/4	8/16-bit
XOR = Exclusive or:						
Reg/memory and register to either	001100dw	mod reg r/m			3/10*	
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0011010w	data	data if w=1		3/4	8/16-bit
NOT = Invert register/memory	1111011w	mod 010 r/m			3/10*	
STRING MANIPULATION						
MOVS = Move byte/word	1010010w				14*	
CMPS = Compare byte/word	1010011w				22*	
SCAS = Scan byte/word	1010111w				15*	
LODS = Load byte/wd to AL/AX	1010110w				12*	
STOS = Store byte/wd from AL/AX	1010101w				10*	
INS = Input byte/wd from DX port	0110110w				14	
OUTS = Output byte/wd to DX port	0110111w				14	
Repeated by count in CX (REP/ REPE/REPZ/REPNE/REPNZ)						
MOVS = Move string	11110010	1010010w			8+8n*	
CMPS = Compare string	1111001z	1010011w			5+22n*	
SCAS = Scan string	1111001z	1010111w			5+15n*	
LODS = Load string	11110010	1010110w			6+11n*	
STOS = Store string	11110010	1010101w			6+9n*	
INS = Input string	11110010	0110110w			8+8n*	
OUTS = Output string	11110010	0110111w			8+8n*	
CONTROL TRANSFER						
CALL = Call:						
Direct within segment	11101000	disp-low	disp-high		19	
Register/memory indirect within segment	11111111	mod 010 r/m			17/27	
Direct intersegment	10011010	segment offset			31	
		segment selector				
Indirect intersegment	11111111	mod 011 r/m	(mod ≠ 11)		54	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
JMP = Unconditional jump:						
Short/long	11101011	disp-low			14	
Direct within segment	11101001	disp-low	disp-high		14	
Register/memory indirect within segment	11111111	mod 100 r/m			11/21	
Direct intersegment	11101010	segment offset			14	
		segment selector				
Indirect intersegment	11111111	mod 101 r/m	(mod ≠ 11)		34	
RET = Return from CALL:						
Within segment	11000011				20	
Within seg adding immed to SP	11000010	data-low	data-high		22	
Intersegment	11001011				30	
Intersegment adding immediate to SP	11001010	data-low	data-high		33	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued)			
INT = Interrupt:			
Type specified	11001101 | type	47	
Type 3	11001100	45	if INT. taken/
INTO = Interrupt on overflow	11001110	48/4	if INT. not taken
IRET = Interrupt return	11001111	28	
BOUND = Detect value out of range	01100010 | mod reg r/m	33–35	
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{TEST}$ = 0
LOCK = Bus lock prefix	11110000	2	
NOP = No Operation	10010000	3	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

REVISION HISTORY

The sections significantly revised since version -001 are:

LCC Contact Diagram	Corrections made to upper address pins.
Pin Description Table	Various descriptions rewritten for clarity.
Interrupt Vector Table	Redrawn for clarity.
ESC Opcode Exception Description	Note added concerning ESC trap.
Oscillator Configurations	Deleted drive of X2 with inverted X1.
RESET Logic	Deleted paragraph concerning setup times for synchronization of multiple processors.
Local Bus Arbitration	Added description of HLDA when a refresh cycle is pending.
Local Bus Controller and Reset	Added description of pullup devices for appropriate pins.
DMA Controller	Added reminder to initialize transfer count registers and pointer registers.
Timers	Added reminder to initialize count registers.
DRAM Refresh Addresses	Refresh address counter described in figure.
D.C. Characteristics	V_{IH2} indicated for SRDY, ARDY. I_{CC} (max.) now indicated for all devices.
Power Supply Current	Typical I_{CC} indicated.
A.C. Characteristics	Input V_{IH} test condition at X1 added. T_{CLDOX}, T_{CVCTV}, T_{CVCTX}, T_{CLHAV} and T_{CLLV} minimums reduced from 5 ns to 3 ns. T_{CLCH} min. and T_{CHCL} min. relaxed by 2 ns. Added reminder that T_{SRYCL} and T_{CLSRY} must be met.
Explanation of the A.C. Symbols	New section.
Major Cycle Timing Waveforms	T_{DXDL} indicated in Read Cycle. T_{CLRO} indicated.
Rise/Fall Times and Capacitive Derating Curves	New Figures added.
Instruction Set Summary	ESC deleted.

82188
INTEGRATED BUS CONTROLLER FOR 8086, 8088, 80186, 80188 PROCESSORS

- **Provides Flexibility in System Configurations**
  - **— Supports 8087 Numerics Coprocessor in 8 MHz 80186 and 80188 Systems**
  - **— Provides a Low-cost Interface for 8086, 8088 Systems to an 82586 LAN Coprocessor or 82730 Text Coprocessor**
- **Facilitates Interface to one or more Multimaster Busses**
- **Supports Multiprocessor, Local Bus Systems**
- **Allows use of 80186/80188 High-Integration Features**
- **3-State, Command Output Drivers**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**
- **Available in Plastic DIP or Cerdip Package**

(See Packaging Outlines and Dimensions, Order #231369)

The 82188 Integrated Bus Controller (IBC) is a 28-pin HMOS III component for use with 80186, 80188, 8086 and 8088 systems. The IBC provides command and control timing signals plus a configurable $\overline{RQ}/\overline{GT} \longleftrightarrow$ HOLD-HLDA converter. The device may be used to interface an 8087 Numerics Coprocessor with an 80186 or 80188 Processor. Also, an 82586 Local Area Network (LAN) Coprocessor or 82730 Text Coprocessor may be interfaced to an 8086 or 8088 with the IBC.

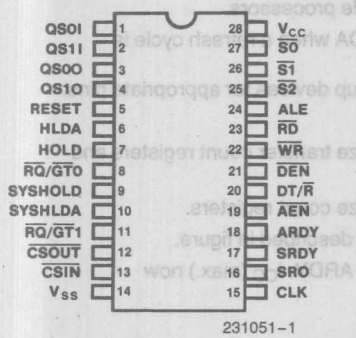


Figure 1.
82188 Pin Configuration

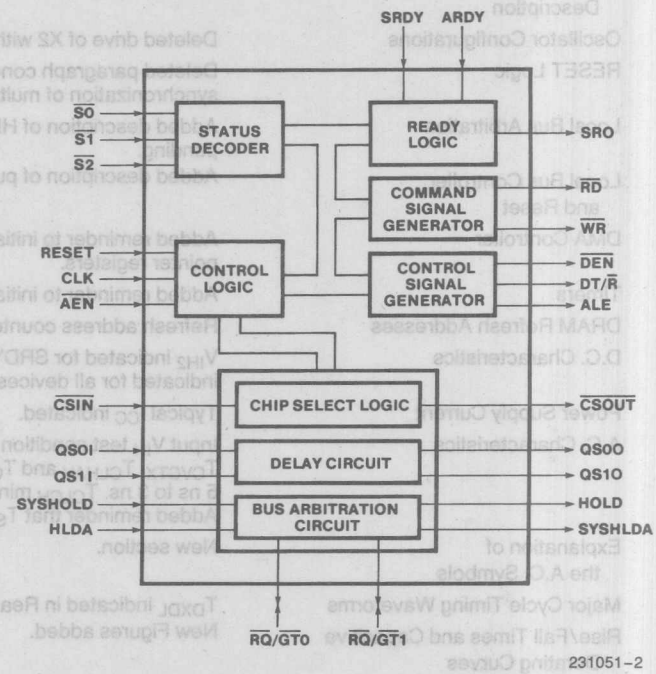


Figure 2.
82188 Block Diagram

September 1988
Order Number: 231051-005

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	27 26 25	I	**Status Input Pins** $\overline{S0}$–$\overline{S2}$ correspond to the status pins of the CPU. The 82188 uses the status lines to detect and identify the processor bus cycles. The 82188 decodes $\overline{S0}$–$\overline{S2}$ to generate the command and control signals. $\overline{S0}$–$\overline{S2}$ are also used to insert 3 wait states into the SRO line during the first 256 80186 bus cycles after RESET. A HIGH input on all three lines indicates that no bus activity is taking place. The status input lines contain weak internal pull-up devices.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read data from memory
1	1	0	write data to memory
1	1	1	passive (no bus cycle)

CLK	15	I	**CLOCK** CLK is the clock signal generated by the CPU or clock generator device. CLK edges establish when signals are sampled and generated.
RESET	5	I	**RESET** RESET is a level triggered signal that corresponds to the system reset signal. The signal initializes an internal bus cycle counter, thus enabling the 82188 to insert internally generated wait states into the SRO signal during system initialization. The 82188 mode is also determined during RESET. $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ are driven HIGH during RESET regardless of $\overline{AEN}$. RESET is active HIGH.
$\overline{AEN}$	19	I	**Address Enable** This signal enables the system command lines when active. If $\overline{AEN}$ is inactive (HIGH), $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ will be tri-stated and ALE will be driven LOW (DT/$\overline{R}$ will not be effected). $\overline{AEN}$ is an asynchronous signal and is active LOW.
ALE	24	O	**Address Latch Enable** This signal is used to strobe an address into address latches. ALE is active HIGH and latch should occur on the HIGH to LOW transition. ALE is intended for use with transparent D-type latches.
$\overline{DEN}$	21	O	**Data Enable** This signal is used to enable data transceivers located on either the local or system data bus. The signal is active LOW. $\overline{DEN}$ is tri-stated when $\overline{AEN}$ is inactive.
DT/$\overline{R}$	20	O	**Data TRANSMIT/RECEIVE** This signal establishes the direction of data flow through the data transceivers. A HIGH on this line indicates TRANSMIT (write to I/O or memory) and a LOW indicates RECEIVE (Read from I/O or memory).

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	23	O	**READ** This signal instructs an I/O or memory device to drive its data onto the data bus. The $\overline{RD}$ signal is similiar to the $\overline{RD}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{RD}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
$\overline{WR}$	22	O	**WRITE** This signal instructs an I/O or memory device to record the data presented on the data bus. The $\overline{WR}$ signal is similiar to the $\overline{WR}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{WR}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
HOLD	7	O	**HOLD** The HOLD signal is used to request bus control from the 80186 or 80188. The request can come from either the 8087 ($\overline{RQ}/\overline{GT}O$) or from the third processor (SYSHOLD). The signal is active HIGH.
HLDA	6	I	**HOLD Acknowledge** 80186 MODE–This line serves to translate the HLDA output of the 80186(80188) to the appropriate signal of the device requesting the bus. HLDA going active (HIGH) indicates that the 80186 has relinquished the bus. If the requesting device is the 8087, HLDA will be translated into the grant pulse of the $\overline{RQ}/\overline{GT}O$ line. If the requesting device is the optional third processor, HLDA will be routed into the SYSHLDA line. This pin also determines the mode in which the 82188 will operate. If this line is HIGH during the falling edge of RESET, the 82188 will enter the 8086 mode. If LOW, the 82188 will enter the 80186 mode. For 8086 mode, this pin should be strapped to V_{CC}.
$\overline{RQ}/\overline{GT}O$	8	I/O	**Request/Grant O** $\overline{RQ}/\overline{GT}O$ is connected to $\overline{RQ}/\overline{GT}O$ of the 8087 Numeric Coprocessor. When initiated by the 8087, $\overline{RQ}/\overline{GT}O$ will be translated to HOLD-HLDA to acquire the bus from the 80186(80188). This line is bidirectional, and is active LOW. $\overline{RQ}/\overline{GT}O$ has a weak internal pull-up device to prevent erroneous request/grant signals.
$\overline{RQ}/\overline{GT}1$	11	I/O	**Request/Grant 1** 80186 Mode–In 80186 Mode, $\overline{RQ}/\overline{GT}1$ allows a third processor to take control of the local bus when the 8087 has bus control. For a HOLD-HLDA type third processor, the 82188's $\overline{RQ}/\overline{GT}1$ line should be connected to the $\overline{RQ}/\overline{GT}1$ line of the 8087. 8086 MODE–In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is connected to either $\overline{RQ}/\overline{GT}O$ or $\overline{RQ}/\overline{GT}1$ of the 8086. $\overline{RQ}/\overline{GT}1$ will start its request/grant sequence when the SYSHOLD line goes active. In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is used to gain bus control from the 8086 or 8088. $\overline{RQ}/\overline{GT}1$ is a bidirectional line and is active LOW. This line has a weak internal pull-up device to prevent erroneous request/grant signals.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
SYSHOLD	9	I	**System Hold** 80186 MODE–SYSHOLD serves as a hold input for an optional third processor in an 80186(80188)-8087 system. If the 80186(80188) has bus control, SYSHOLD will be routed to HOLD to gain control of the bus. If the 8087 has bus control, SYSHOLD will be translated to $\overline{RQ}/\overline{GT}1$ to gain control of the bus. 8086 MODE–SYSHOLD serves as a hold input for a coprocessor in an 8086 or 8088 system. SYSHOLD is translated to $\overline{RQ}/\overline{GT}1$ of the 82188 to allow the coprocessor to take control of the bus. SYSHOLD may be an asynchronous signal.
SYSHLDA	10	O	**System Hold Acknowledge** SYSHLDA serves as a hold acknowledge line to the processor or coprocessor connected to it. The device connected to the SYSHOLD-SYSHLDA lines is allowed the bus when SYSHLDA goes active (HIGH).
SRDY	17	I	**Synchronous Ready** The SRDY input serves the same function as SRDY of the 80186(80188). The 82188 combines SRDY with ARDY to form a synchronized ready output signal (SRO). SRDY must be synchronized external to the 82188 and is active HIGH. If tied to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 cycles following RESET. If only ARDY is to be used, SRDY should be tied LOW.
ARDY	18	I	**Asynchronous Ready** The ARDY input serves the same function as ARDY of the 80186(80188). ARDY may be an asynchronous input, and is active HIGH. Only the rising edge of ARDY is synchronized by the 82188. The falling edge must be synchronized external to the 82188. If connected to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 bus cycles following RESET. If only SRDY is to be used, ARDY should be connected LOW.
SRO	16	O	**Synchronous READY Output** SRO provides a synchronized READY signal which may be interfaced directly with the SRDY of the 80186(80188) and READY of the 8087. The SRO signal is an accumulation of the synchronized ARDY signal, the SRDY signal, and the internally generated wait state signal.
QS0I QS1I	1 2	I	**Queue-Status Inputs** QS0I, QS1I are connected to the Queue-Status lines of the 80186(80188) to allow synchronization of the queue-status signals to 8087 timing requirements.
QS0O QS1O	3 4	O	**Queue-Status Outputs** QS0O, QS1O are connected to the queue-status pins of the 8087. The signals produced meet 8087 Queue-Status input requirements.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{CSIN}$	13	I	**Chip-Select Input** $\overline{CSIN}$ is connected to one of the chip-select lines of the 80186(80188). $\overline{CSIN}$ informs the 82188 that a bank select is taking place. The 82188 routes this signal to the chip-select output ($\overline{CSOUT}$). $\overline{CSIN}$ is active LOW. This line is not used when memory and I/O device addresses are decoded external to the 80186(80188).
$\overline{CSOUT}$	12	O	**Chip-Select Output** This signal is used as a chip-select line for a bank of memory devices. It is active when $\overline{CSIN}$ is active or when the 8087 has bus control. $\overline{CSOUT}$ is active LOW.

FUNCTIONAL DESCRIPTION

BUS CONTROLLER

The 82188 Integrated Bus Controller (IBC) generates system control and command signals. The signals generated are determined by the Status Decoding Logic. The bus controller logic interprets status lines $\overline{S0}$–$\overline{S2}$ to determine what type of bus cycle is taking place. The appropriate signals are then generated by the Command and Control Signal Generators.

The Address Enable ($\overline{AEN}$) line allows the command and control signals to be disabled. When $\overline{AEN}$ is inactive (HIGH), the command signals and $\overline{DEN}$ will be tri-stated, and ALE will be held low (DT/$\overline{R}$ will be uneffected). $\overline{AEN}$ inactive will allow other systems to take control of the bus. Control and command signals respond to a change in the $\overline{AEN}$ signal within 40 ns.

The command signals consist of $\overline{RD}$ and $\overline{WR}$. The 82188's $\overline{RD}$ and $\overline{WR}$ signals are similiar to $\overline{RD}$ and $\overline{WR}$ of the 80186(80188) in the non-Queue-Status Mode. These command signals do not differentiate between memory and I/O devices. $\overline{RD}$ and $\overline{WR}$ can be conditioned by $\overline{S2}$ of the 80186(80188) to obtain separate signals for I/O and memory devices. $\overline{RD}$ is asserted during INTA cycles, unlike $\overline{RD}$ on the 80186(80188).

The control commands consist of Data Enable ($\overline{DEN}$), Data Transmit/Receive (DT/$\overline{R}$), and Address Latch Enable (ALE). The control commands are similiar to those generated by the 80186(80188). $\overline{DEN}$ determines when the external bus should be enabled onto the local bus. DT/$\overline{R}$ determines the direction of the data transfer, and ALE determines when the address should be strobed into the latches (used for demultiplexing the address bus). DT/$\overline{R}$ does not go to an inactive (high) state at the end of bus cycles, unlike DT/$\overline{R}$ on the 80186(80188).

MODE SELECT

The 82188 Integrated Bus Controller (IBC) is configurable. The device has two modes: 80186 Mode and 8086 Mode. Selecting the mode of the device configures the Bus Arbitration Logic (see BUS ARBITRATION section for details). In 80186 Mode, the 82188 IBC may be used as a bus controller/interface device for an 80186(80188), 8087, and optional third processor system. In 8086 Mode, the 82188 IBC may be used as an interface device allowing a maximum mode 8086(8088) to interface with a coprocessor that uses a HOLD-HLDA bus exchange protocol.

The mode of the 82188 is determined during RESET. If the HLDA line is LOW at the falling edge of RESET (as in the case when tied to the HLDA line of the 80186 or 80188), the 82188 will enter into 80186 Mode. If the HLDA line is HIGH at the falling edge of RESET, the 82188 will enter 8086 Mode. In 8086 Mode, only the Bus Arbitration Logic is used. The eight pins used in 8086 Mode are: SYSHOLD, SYSHLDA, HLDA, CLK, RESET, $\overline{RQ}/\overline{GT}1$, V_{CC}, and V_{SS}. The other pins may be left unconnected.

BUS ARBITRATION

The Bus Exchange Logic interfaces up to three sets of bus exchange signals:

- HOLD-HLDA
- SYSHOLD-SYSHLDA
- $\overline{RQ}/\overline{GT}0$ ($\overline{RQ}/\overline{GT}1$)

This logic executes translating, routing, and arbitrating functions. The logic translates HOLD-HLDA signals to $\overline{RQ}/\overline{GT}$ signals and $\overline{RQ}/\overline{GT}$ signals to HOLD-HLDA signals. The logic also determines which set of bus exchange signals are to be interfaced. The mode of the 82188 and the priority of the devices requesting the bus determine the routing of the bus exchange signals.

80186 MODE

In 80186 Mode, a system may have three potential bus masters: the 80186 or 80188 CPU, the 8087 Numerics Coprocessor, and a third processor (such as the 82586 LAN or 82730 Text Coprocessor). The third processor may have either a HOLD-HLDA or $\overline{RQ}/\overline{GT}$ bus exchange protocol. The possible bus exchange signal connections and paths for 80186 Mode are shown in Figures 3 & 4 and Tables 1 & 2, respectively. If no HOLD-HLDA type third processor is used, SYSHOLD should be tied LOW to prevent an erroneous SYSHOLD signal. In 80186 mode, the bus priorities are:

Highest Priority . . . . . Third Processor
Second Highest Priority 8087
Default Priority 80186

— THREE-PROCESSOR SYSTEM OPERATION (HOLD-HLDA TYPE THIRD PROCESSOR)

In the configuration shown in Figure 3, the third processor requests the bus by sending SYSHOLD HIGH. The 82188 will route (and translate if necessary) the request to the current bus master. This includes routing the request to HOLD if the 80186(80188) is the current bus master or routing and translating the request to $\overline{RQ}/\overline{GT}1$ if the 8087 is in control of the bus. The third processor's request is not passed through the 8087 if the 80186 is the bus master (see Table 1).

The 8087 requests the bus using $\overline{RQ}/\overline{GT}0$. The request pulse from the 8087 will be translated and routed to HOLD if the 80186 is the bus master. If the third processor has control of the bus, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus (sending SYSHOLD LOW). In this case, HOLD will remain HIGH during the third processor-to-8087 bus control transfer. The 80186 will not be granted the bus until both coprocessors have released it.

Table 1. Bus Exchange Paths (80186 Mode) (HOLD-HLDA Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \frac{HOLD}{HLDA}$	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \overline{RQ}/\overline{GT}1$	n/a

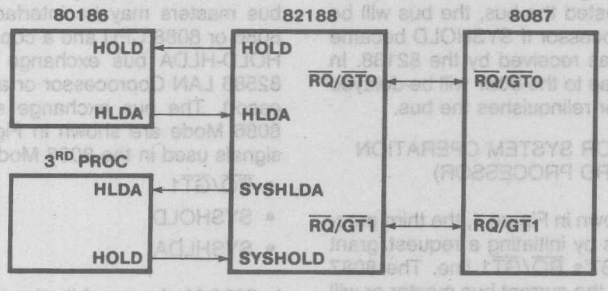


Figure 3.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System (HOLD-HLDA Type 3rd Proc)

Table 2. Bus Exchange Paths (80186 Mode) ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\overline{RQ}/\overline{GT}1 \longleftrightarrow \overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	$\overline{RQ}/\overline{GT}1$	n/a

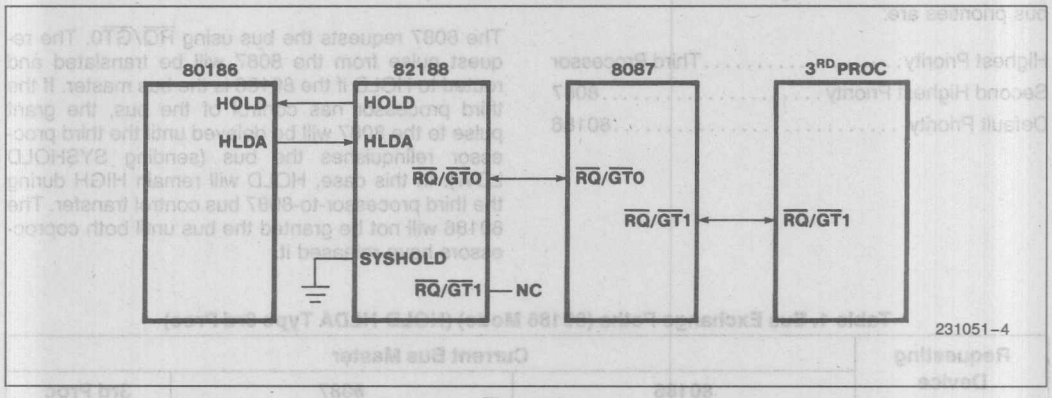


Figure 4.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

When the bus is requested from the 80186(80188), a bus priority decision is made. This decision is made when the HLDA line goes active. Upon receipt of the HLDA signal, the highest-priority requesting device will be acknowledged the bus. For example, if the 8087 initially requested the bus, the bus will be granted to the third processor if SYSHOLD became active before HLDA was received by the 82188. In this case, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus.

— THREE-PROCESSOR SYSTEM OPERATION ($\overline{RQ}/\overline{GT}$ TYPE THIRD PROCESSOR)

In the configuration shown in Figure 4, the third processor requests the bus by initiating a request/grant sequence with the 8087's $\overline{RQ}/\overline{GT}1$ line. The 8087 will grant the bus if it is the current bus master or will pass the request on if the 80186 is the current bus master (see Table 2). In this configuration, the 82188's Bus Arbitration Logic translates $\overline{RQ}/\overline{GT}0$ to HOLD-HLDA. The 8087 provides the bus arbitration in this configuration.

8086 MODE

The 8086 Mode allows an 8086, 8088 system to contain both $\overline{RQ}/\overline{GT}$ and HOLD-HLDA type coprocessors simultaneously. In 8086 Mode, two possible bus masters may be interfaced by the 82188; an 8086 or 8088 CPU and a coprocessor which uses a HOLD-HLDA bus exchange protocol (typically an 82586 LAN Coprocessor or an 82730 Text Coprocessor). The bus exchange signal connections for 8086 Mode are shown in Figure 5. Bus arbitration signals used in the 8086 Mode are:

- $\overline{RQ}/\overline{GT}1$
- SYSHOLD
- SYSHLDA

In 8086 Mode, no arbitration is necessary since only two devices are interfaced. The coprocessor has bus priority over the 8086(8088). SYSHOLD-SYSHLDA are routed and translated directly to $\overline{RQ}/\overline{GT}1$. $\overline{RQ}/\overline{GT}1$ of the 82188 may be tied to either $\overline{RQ}/\overline{GT}0$ or $\overline{RQ}/\overline{GT}1$ of the 8086(8088).

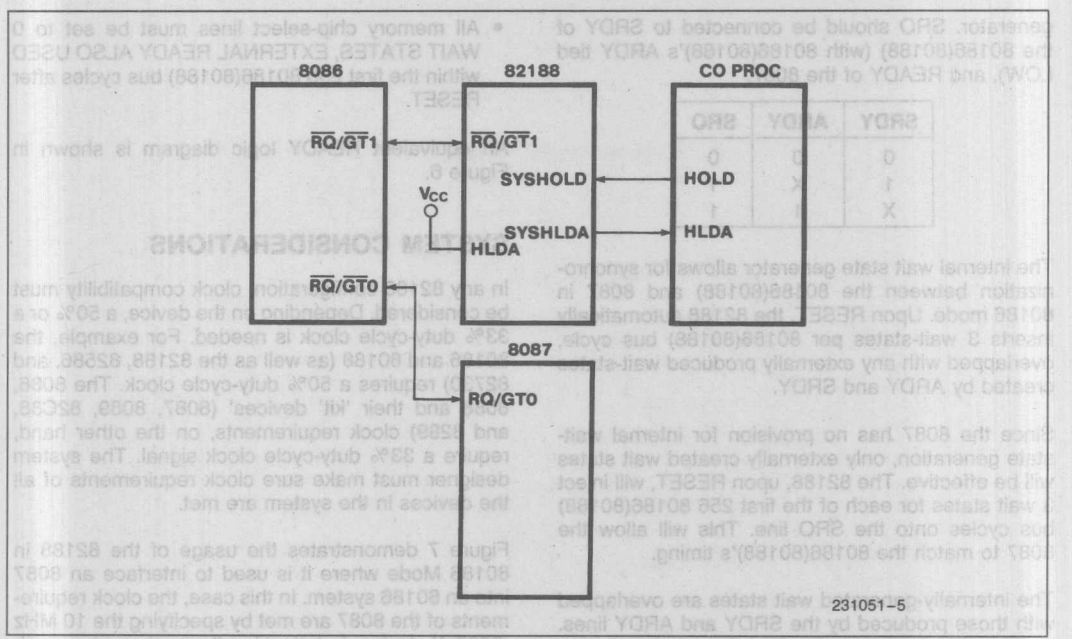


Figure 5. Bus Exchange Signal Connections (8086 Mode)

QUEUE-STATUS DELAY

The Queue-Status Delay logic is used to delay the queue-status signals from the 80186(80188) to meet 8087 queue-status timing requirements. QS0I, QS1I correspond to the queue-status lines of the 80186(80188). The 82188 delays these signals by one clock phase. The delayed signals are interfaced to the 8087 queue-status lines by QS0O, QS1O.

CHIP-SELECT

The Chip-Select Logic allows the utilization of the chip select circuitry of the 80186(80188). Normally, this circuitry could not be used in an 80186(80188)-8087 system since the 8087 contains no chip select circuitry. The Chip-Select Logic contains two external connections: Chip-Select Input ($\overline{CSIN}$) and Chip-Select Output ($\overline{CSOUT}$). $\overline{CSOUT}$ is active when either $\overline{CSIN}$ is active or when the 8087 has control of the bus.

By using $\overline{CSOUT}$ to select memory containing data structures, no external decoding is necessary. The 80186 may gain access to this memory bank through the $\overline{CSIN}$ line while the 8087 will automatically obtain access when it becomes the bus master. Note that this configuration limits the amount of memory accessible by the 8087 to the physical memory bank selected by $\overline{CSOUT}$. Systems where the 8087 must access the full 1 Megabyte address space must use an external decoding scheme.

READY

The Ready logic allows two types of Ready signals: a Synchronous Ready Signal (SRDY) and an Asynchronous Ready Signal (ARDY). These signals are similiar to SRDY and ARDY of the 80186. Wait states will be inserted when both SRDY and ARDY are LOW. Inserting wait states allows slower memory and I/O devices to be interfaced to the 80186(80188)-8087 system.

ARDY's LOW-to-HIGH transition is synchronized to the CPU clock by the 82188. The 82188 samples ARDY at the beginning of T2, T3 and Tw until sampled HIGH. Note that ARDY of the 82188 is sampled one phase earlier than ARDY of the 80186. ARDY's falling edge must be synchronous to the CPU clock. ARDY allows an easy interface with devices that emit an asynchronous ready signal.

The SRDY signal allows direct interface to devices that emit a synchronized ready signal. SRDY must be synchronized to the CPU clock for both of its transitions. SRDY is sampled in the middle of T2, T3 and in the middle of each Tw. An 82188-80186(80188)'s SRDY setup time is 30 ns longer than the 80186(80188)'s SRDY setup time. SRDY eliminates the half-clock cycle penalty necessary for ARDY to be internally sychronized.

The sychronized ready output (SRO) is the accumulation of SRDY, ARDY, and the internal wait-state

generator. SRO should be connected to SRDY of the 80186(80188) (with 80186(80188)'s ARDY tied LOW), and READY of the 8087.

SRDY	ARDY	SRO
0	0	0
1	X	1
X	1	1

The internal wait state generator allows for synchronization between the 80186(80188) and 8087 in 80186 mode. Upon RESET, the 82188 automatically inserts 3 wait-states per 80186(80188) bus cycle, overlapped with any externally produced wait-states created by ARDY and SRDY.

Since the 8087 has no provision for internal wait-state generation, only externally created wait states will be effective. The 82188, upon RESET, will inject 3 wait states for each of the first 256 80186(80188) bus cycles onto the SRO line. This will allow the 8087 to match the 80186(80188)'s timing.

The internally-generated wait states are overlapped with those produced by the SRDY and ARDY lines. Overlapping the injected wait states insures a minimum of three wait states for the first 256 80186(80188) bus cycles after RESET. Systems with a greater number of wait states will not be affected. Internal wait state generation by the 82188 will stop on the 256th 80186(80188) bus cycle after RESET. To maintain sychronization between the 80186(80188) and 8087, the following conditions are necessary:

- The 80186(80188)'s control block must be mapped in I/O space before it is written to or read from.
- All memory chip-select lines must be set to 0 WAIT STATES, EXTERNAL READY ALSO USED within the first 256 80186(80188) bus cycles after RESET.

An equivalent READY logic diagram is shown in Figure 6.

SYSTEM CONSIDERATIONS

In any 82188 configuration, clock compatibility must be considered. Depending on the device, a 50% or a 33% duty-cycle clock is needed. For example, the 80186 and 80188 (as well as the 82188, 82586, and 82730) requires a 50% duty-cycle clock. The 8086, 8088 and their 'kit' devices' (8087, 8089, 82C88, and 8289) clock requirements, on the other hand, require a 33% duty-cycle clock signal. The system designer must make sure clock requirements of all the devices in the system are met.

Figure 7 demonstrates the usage of the 82188 in 80186 Mode where it is used to interface an 8087 into an 80186 system. In this case, the clock requirements of the 8087 are met by specifying the 10 MHz (8087-1) device, but clocking the system at a maximum rate of 8 MHz.

Status bit six (S6) from the main processor (8086, 8088, 80186, or 80188) is used by the 8087 to track the instruction flow. S6 is multiplexed with address bit 19 (A19). If the third processor generates only 16 bits of address, S6 is not generated. A19/S6 must be driven high by external circuitry during the status portion of bus cycles controlled by the third processor.

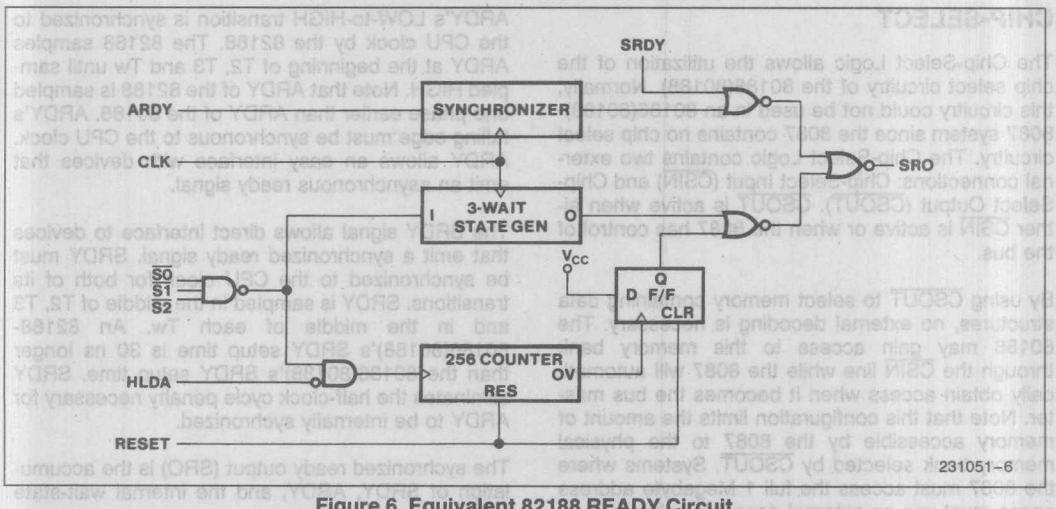


Figure 6. Equivalent 82188 READY Circuit

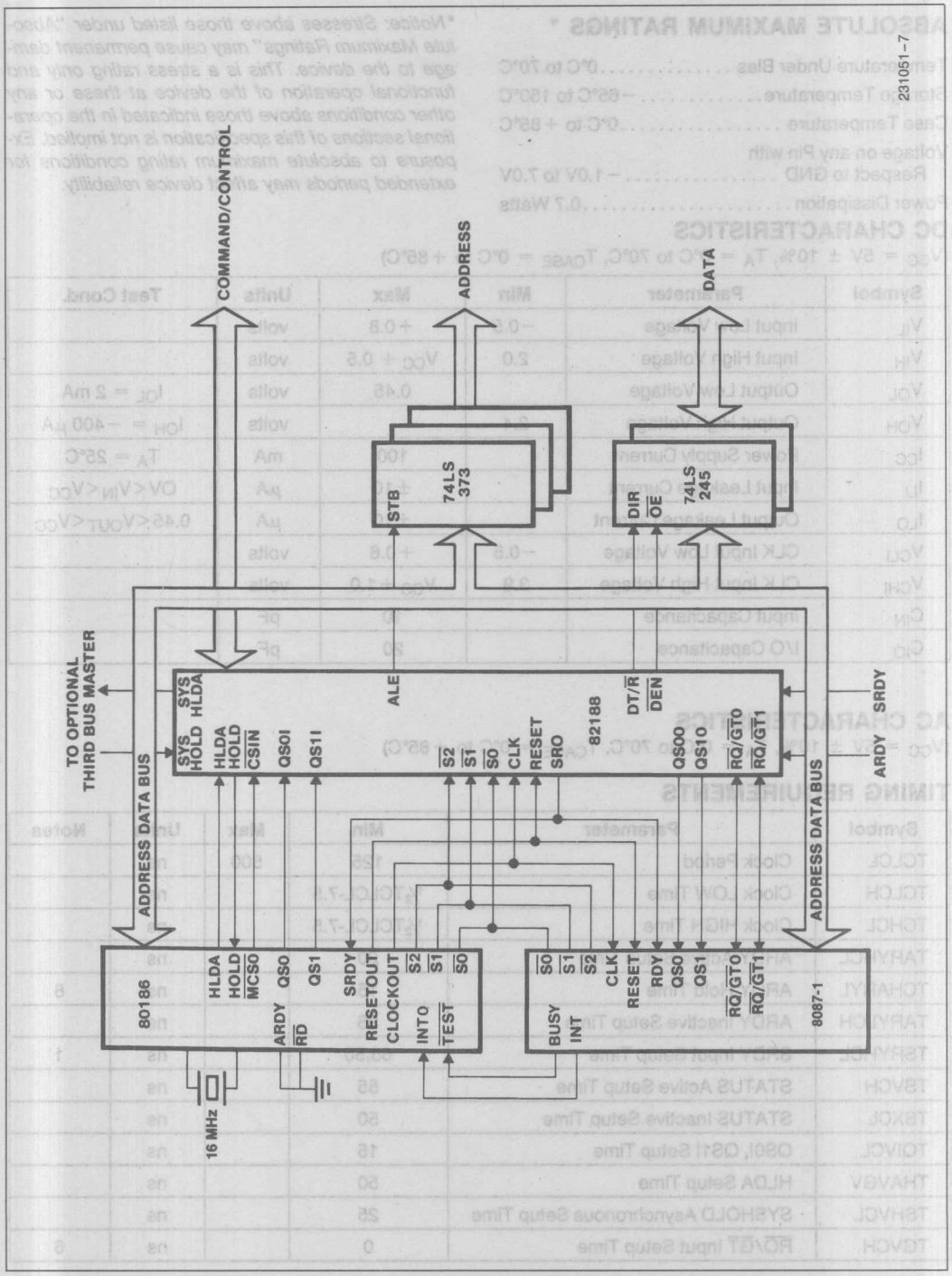


Figure 7.
80186/8087-1 System Using the 82188 in 80186 Mode

ABSOLUTE MAXIMUM RATINGS *

Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Case Temperature ..... 0°C to +85°C
Voltage on any Pin with Respect to GND ..... −1.0V to 7.0V
Power Dissipation ..... 0.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

Symbol	Parameter	Min	Max	Units	Test Cond.
V_{IL}	Input Low Voltage	−0.5	+0.8	volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	volts	
V_{OL}	Output Low Voltage		0.45	volts	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		100	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$OV < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45 < V_{OUT} < V_{CC}$
V_{CLI}	CLK Input Low Voltage	−0.5	+0.6	volts	
V_{CHI}	CLK Input High Voltage	3.9	V_{CC} +1.0	volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

AC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

TIMING REQUIREMENTS

Symbol	Parameter	Min	Max	Units	Notes
TCLCL	Clock Period	125	500	ns	
TCLCH	Clock LOW Time	½TCLCL-7.5		ns	
TCHCL	Clock HIGH Time	½TCLCL-7.5		ns	
TARYHCL	ARDY Active Setup Time	20		ns	
TCHARYL	ARDY Hold Time	15		ns	8
TARYLCH	ARDY Inactive Setup Time	35		ns	
TSRYHCL	SRDY Input Setup Time	65,50		ns	1
TSVCH	STATUS Active Setup Time	55		ns	
TSXCL	STATUS Inactive Setup Time	50		ns	
TQIVCL	QS0I, QS1I Setup Time	15		ns	
THAVGV	HLDA Setup Time	50		ns	
TSHVCL	SYSHOLD Asynchronous Setup Time	25		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Input Setup Time	0		ns	6

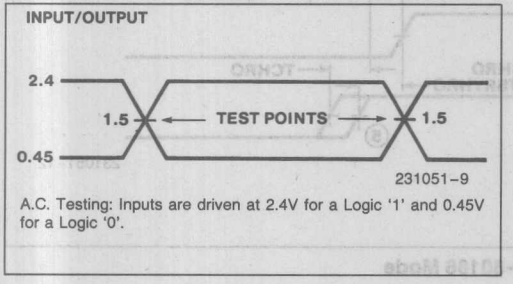
TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Notes
TSVLH	STATUS Valid to ALE Delay		30	ns	4
TCHLL	ALE Inactive Delay		30	ns	
TCLML	$\overline{RD}$, $\overline{WR}$ Active Delay	10	70	ns	
TCLMH	$\overline{RD}$, $\overline{WR}$ Inactive Delay	10	55	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay		30	ns	3
TCLDTV	DT/$\overline{R}$ Active Delay		55	ns	3
TCHDNV	$\overline{DEN}$ Active Delay	10	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay	10	55	ns	
TCLQOV	QS0O, QS1O Delay	5	50	ns	
TCHHV	HOLD Delay		50	ns	2,6
TCLSAV	SYSHLDA Delay		50	ns	6
TCLGV	$\overline{RQ}$/$\overline{GT}$ Output Delay		40	ns	6
TGVHV	$\overline{RQ}$/$\overline{GT}$0 To HOLD Delay		50	ns	2,6
TCLLH	ALE Active Delay		30	ns	4
TAELCV	Command Enable Delay		40	ns	
TAEHCX	Command Disable Delay		40	ns	
TCHRO	SRO Output Delay	5	30	ns	5,6
TSRYHRO	SRDY To SRO Delay		30	ns	5
TCSICSO	$\overline{CSIN}$ To $\overline{CSOUT}$ Delay		30	ns	
TCLCSOV	CLK Low to $\overline{CSOUT}$ Delay	10		ns	
TCLCSOH	CLK Low to $\overline{CSOUT}$ Inactive Delay	10		ns	

NOTES (applicable to both spec listing and timing diagrams):

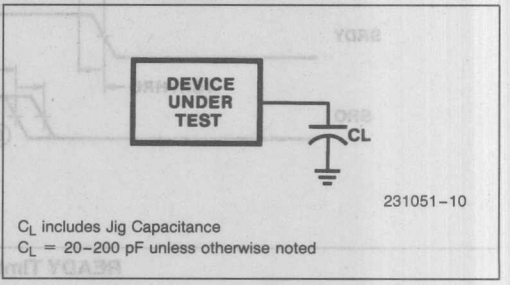
1. TSRYHOL = (80186's) TSRYCL + 30 ns = 65 ns for 6 MHz operation and 50 ns for 8 MHz operation.
2. Timing not tested.
3. DT/$\overline{R}$ will be asserted to the latest of TSVDTV & TCLDTV.
4. ALE will be asserted to the latest of TSVLH & TCLLH.
5. SRO will be asserted to the latest of TCHRO & TSRYHRO.
6. CL = 20–100 pF
7. Address/Data bus shown for reference only.
8. The falling edge of ARDY must be synchronized to CLK.

A.C. TESTING INPUT, OUTPUT WAVEFORM

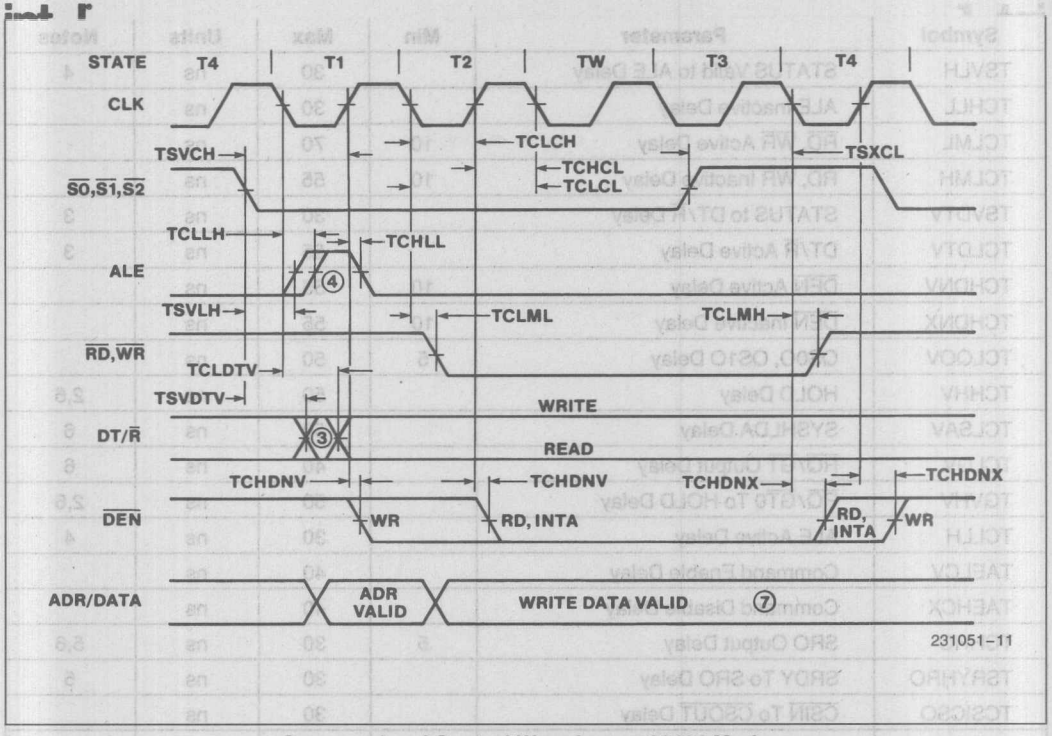


A.C. Testing: Inputs are driven at 2.4V for a Logic '1' and 0.45V for a Logic '0'.

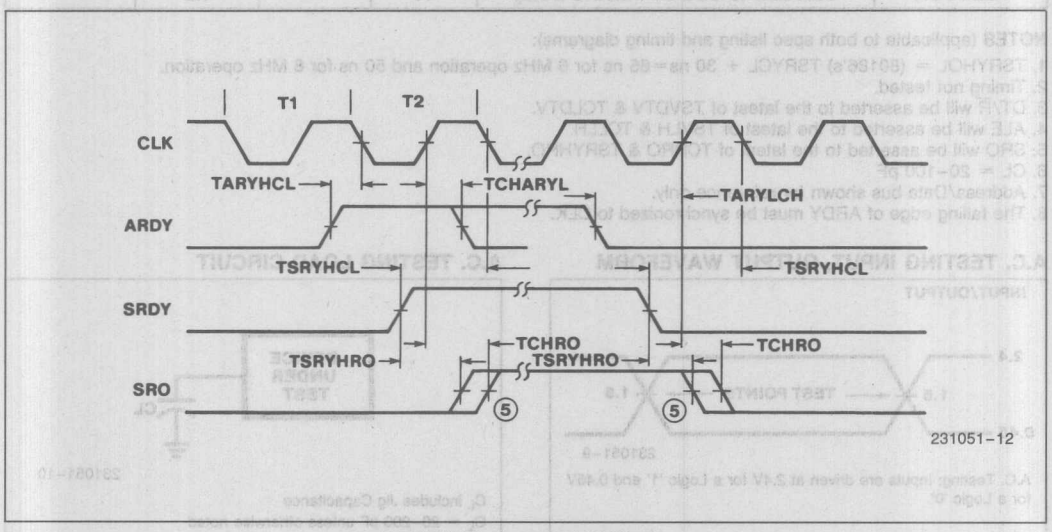
A.C. TESTING LOAD CIRCUIT



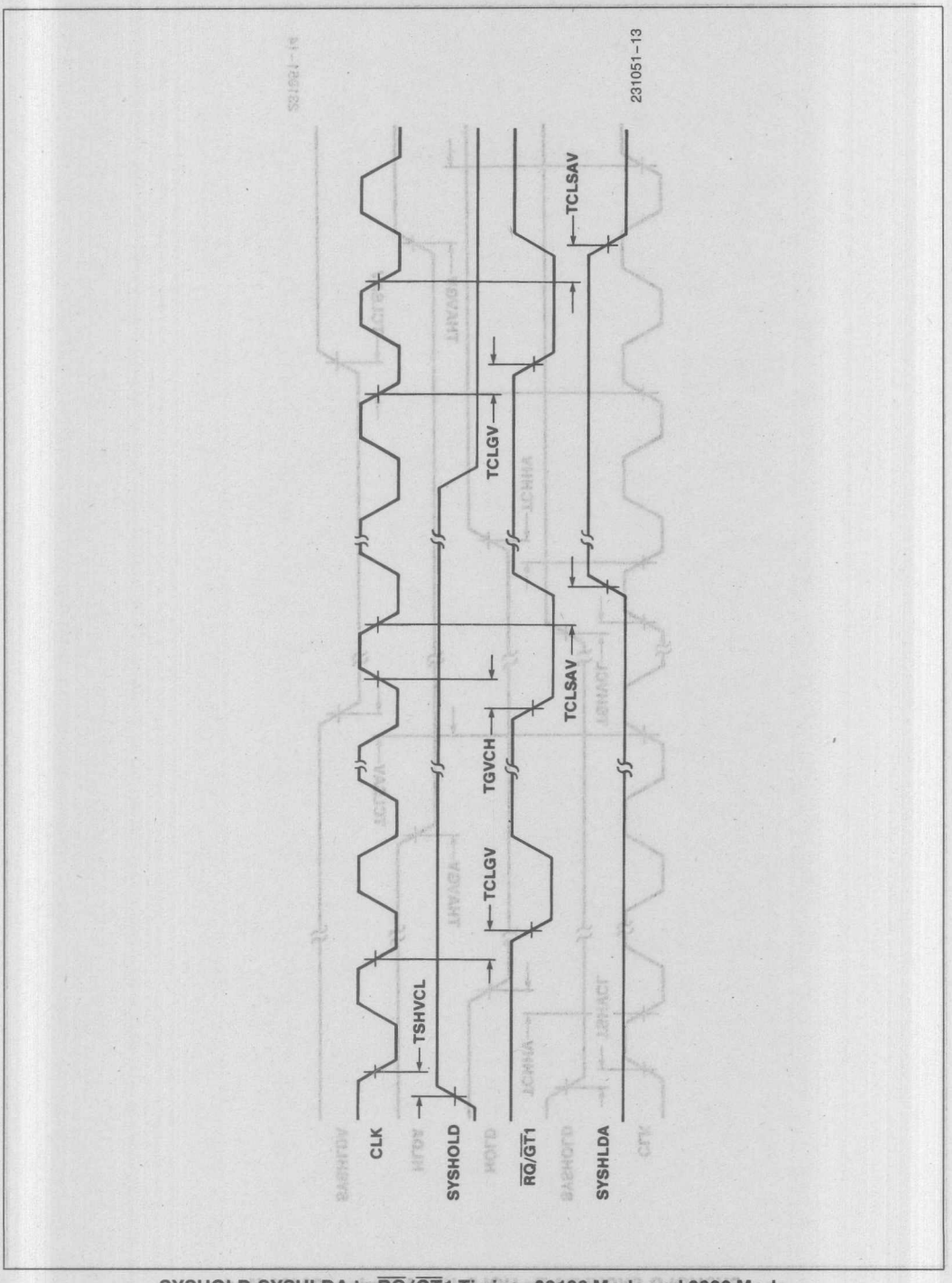
C_L includes Jig Capacitance
C_L = 20–200 pF unless otherwise noted



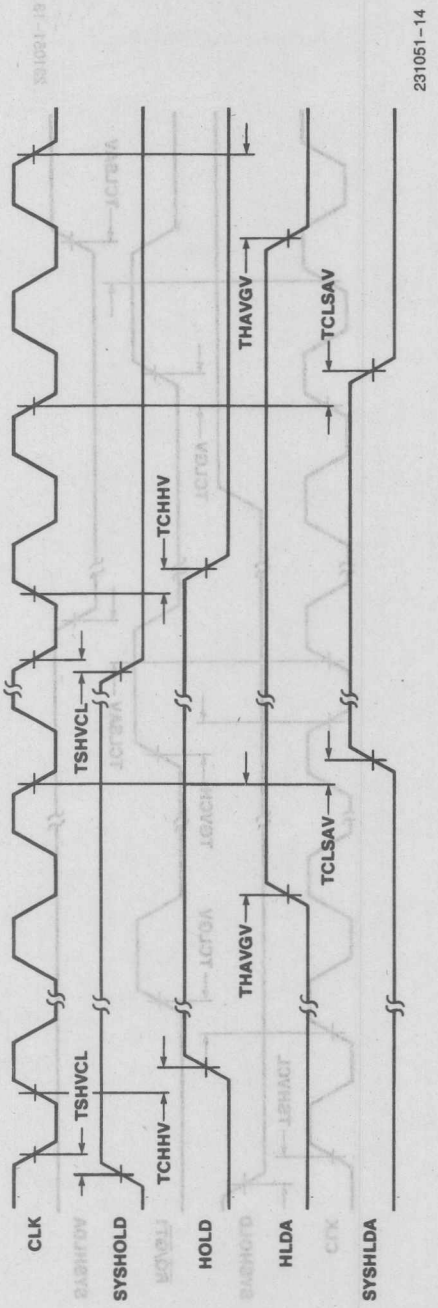
Command and Control Waveforms–80186 Mode



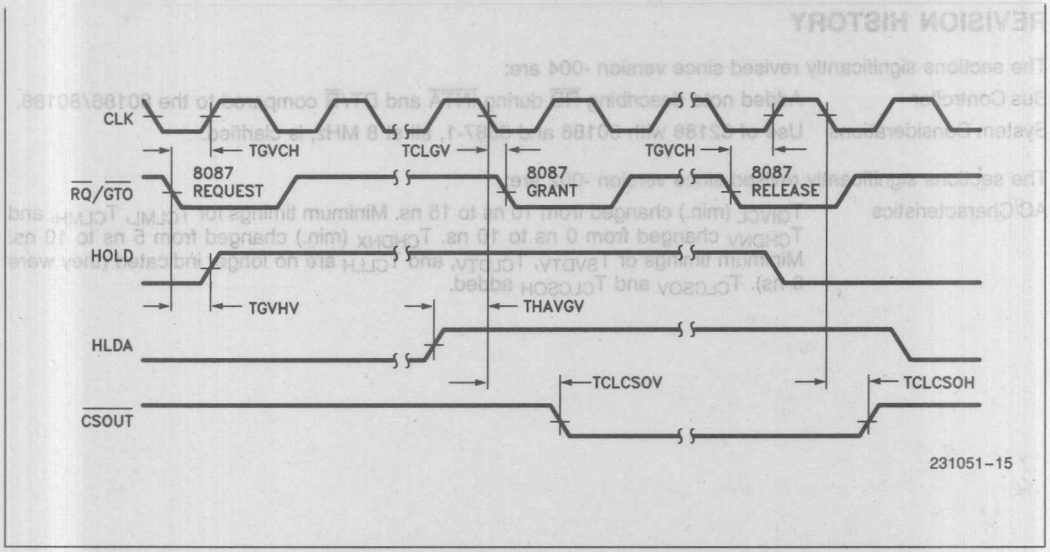
READY Timing–80186 Mode



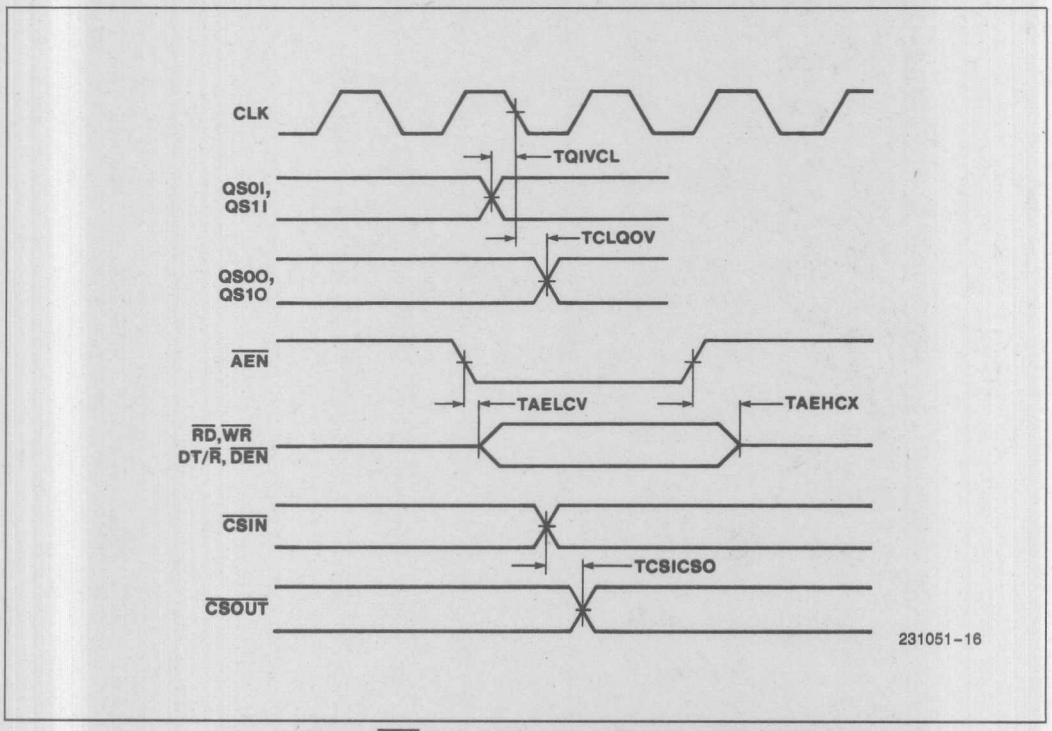
SYSHOLD-SYSHLDA to $\overline{RQ}/\overline{GT1}$ Timing–80186 Mode and 8086 Mode



SYSHOLD-SYSHLDA To HOLD-HLDA Timing–80186 Mode



RQ/GT0 to HOLD-HLDA Timing–80186 Mode



Queue Status, ALE, Chip Select Delay Timing–80186 Mode

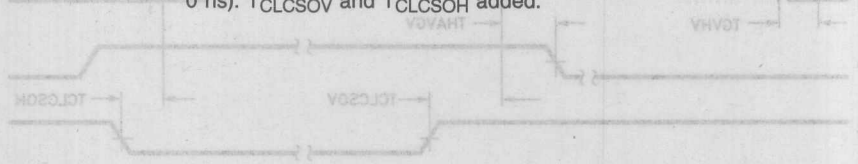
REVISION HISTORY

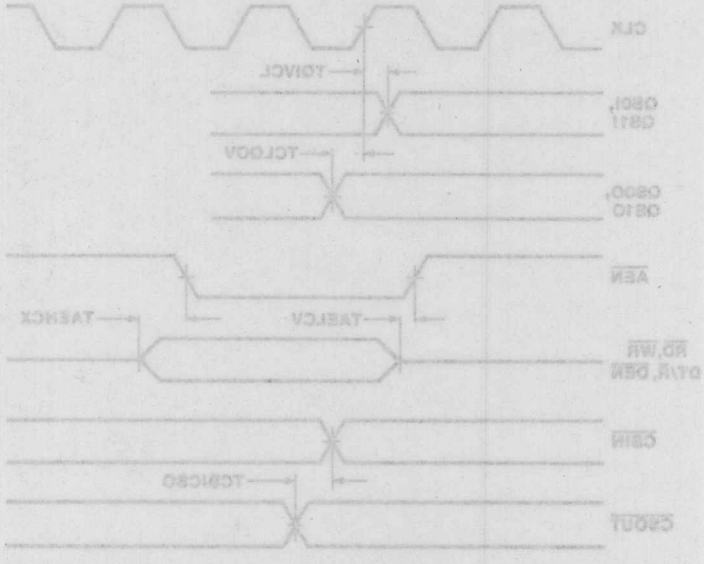
The sections significantly revised since version -004 are:

Bus Controller	Added note describing $\overline{RD}$ during $\overline{INTA}$ and DT/$\overline{R}$ compared to the 80186/80188.
System Considerations	Use of 82188 with 80186 and 8087-1, all at 8 MHz, is clarified.

The sections significantly revised since version -002 are:

AC Characteristics	T_{QIVCL} (min.) changed from 10 ns to 15 ns. Minimum timings for T_{CLML}, T_{CLMH}, and T_{CHDNV} changed from 0 ns to 10 ns. T_{CHDNX} (min.) changed from 5 ns to 10 ns. Minimum timings or T_{SVDTV}, T_{CLDTV}, and T_{CLLH} are no longer indicated (they were 0 ns). T_{CLCSOV} and T_{CLCSOH} added.





87C75PF
MICROCONTROLLER PERIPHERAL I/O PORT EXPANDER WITH 32Kx8 EPROM

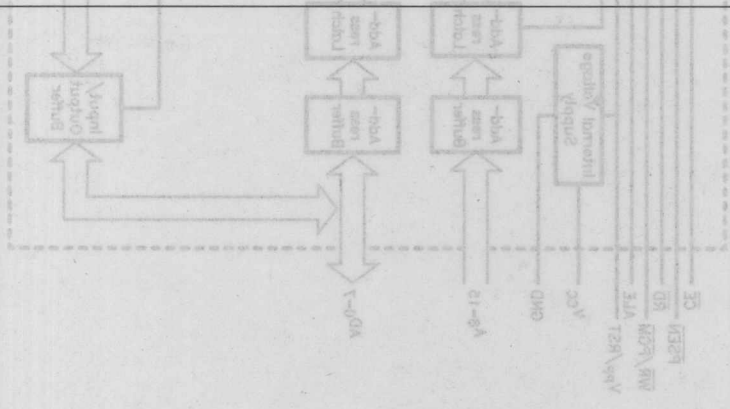
- **2 Configurable 8-bit I/O Ports**
  - **— Open Drain**
  - **— Quasi-bi-directional**
  - **— CMOS**
- **32K x 8 EPROM**
  - **— 200nS Access Time**
- **Quick-Pulse Programming™ Algorithm**
  - **— 4 Second Programming**
- **Configuration Registers**
  - **— Relocate the EPROM in Memory**
  - **— Relocate the SFRs in Memory**
  - **— Programmable RESET Level**
  - **— Double or Single Plane Operation**
- **No-Glue Microcontroller Interface**
  - **— Programmable Memory Map**
  - **— Programmable Control Signals**
  - **— Built-in Address Latches**
  - **— Integrated Address Decoder**
- **Special Function Registers (SFRs)**
  - **— Port Latch Read/Write**
  - **— Port Pin Read**
- **Low Power CHMOS-II-E**
  - **— TTL Compatible**
- **40-Pin DIP, 44-Lead PLCC**
  (See Packaging Spec., Order #231369)

The microcontroller peripheral Port Expander contains two 8-bit bi-directional I/O ports, a 32K x 8 EPROM, fully multiplexed address/data pins, and a user-configurable architecture. A microcontroller that accesses external memory must use two of its 8-bit I/O ports for multiplexed address/data lines. The Port Expander recovers these two ports while supplying needed EPROM memory. Considerable board space and design time can be saved by replacing discrete memory, port, address-decoder, address-latch, and glue chips with a single Port Expander chip.

User-programmable options allow "no-glue" interfacing with 8051, 8096, and 80188 microcontroller families. EPROM and port addresses can be relocated within dual-64K-byte memory planes. Non-standard-architecture microcontrollers (68xx, 63xx, Z8xx, etc.) require only minimal "glue" chips to interface with the Port Expander. The programmable RESET input will conform to various microcontrollers. Its flexible architecture allows applications to use multiple Port Expander chips.

The device's flexibility accommodates several microcontroller architectures. Its default mode is ideal for dual-memory-plane 8051 applications. A single plane option conforms to 80188, 68xx, and 8-bit-mode 8096 architectures. The memory-plane overlap option allows address-constrained systems and 8051 systems that have code compiled from high-level languages to use multiple Port Expanders.

*CHMOS is a patented process of Intel Corporation.



October 1988
Order Number: 290165-002

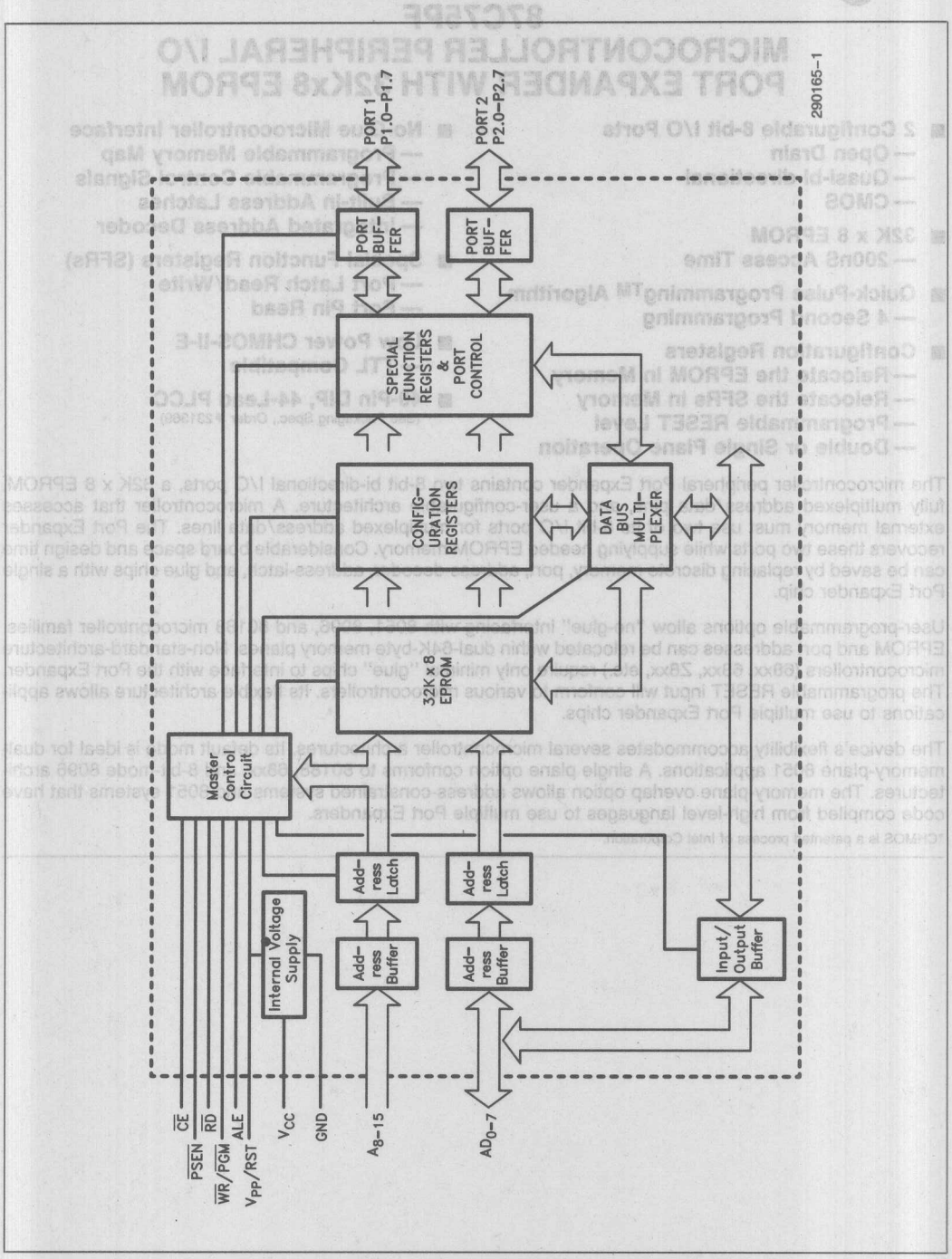


Figure 1. Block Diagram

ARCHITECTURE

Intel's 8051-family and the 87C75PF form the most versatile, integrated microcontroller combination in the industry. No other solution provides a microcontroller, 32K-bytes of EPROM and port expansion in only two chips. Also, the 87C75PF takes full advantage of the 8051's separate program- and data-memory planes. The 87C75PF uses all sixteen address/data lines and all of the 8051's control signals to access two 64K-byte memory planes. In fact, this architecture accommodates two 87C75PFs — 64K-bytes of EPROM and 4 ports — still leaving room for 60K of RAM and other features.

The 87C75PF's versatility makes possible minimum-chip solutions for other microcontroller architectures, too. Single memory-plane modes are user programmable for no-glue interfaces to 8096BH, 80C196, 8098, and 80188 controllers.

Flexible Memory Map

Programmable memory map options will customize the 87C75PF for any application. Intel's 8051 and 8096 microcontrollers have boot-up locations in the lower half of their memory maps. The 87C75PF's EPROM defaults to low memory for these controllers. 80188, 68xx, and 63xx microcontrollers use high-memory boot-up (code and vector) addresses. A user programmable option will move the 87C75PF's EPROM to the device's high-memory addresses. Special Function Registers and port addresses can also be moved to any 2K-byte address boundary.

Programmable Control

Reset level varies depending on the microcontroller family. The 87C75PF's reset (RST) is active-high to match the 8051. Other microcontrollers have active-low reset. A programmable active-low reset option will configure the 87C75PF for these controllers.

Versatile I/O Ports

The 87C75PF has two 8-bit I/O ports. Port 1 is open-drain and port 2 is quasi-bi-directional. The open-drain port can be used for high impedance inputs or "wire-ORed" input/outputs. The quasi-bi-directional port can be used as inputs with built-in pull-up resistors or as low-current-drive outputs. Alternate modes allow either port to have active pull-up (CMOS) outputs. This output mode provides higher current, faster switching, and low power port drive.

Minimum Chip Microcontroller Solution

Primary applications are: 1) single-chip microcontroller systems that have outgrown the controller's internal code-memory and 2) multiple-chip systems that need features-integration, such as redesigned applications that recover ports with discrete components. Typical memory expansion requires EPROM, port expander chips, address latches, address decoder and glue-logic chips — all are incorporated in the 87C75PF (Figure 1).

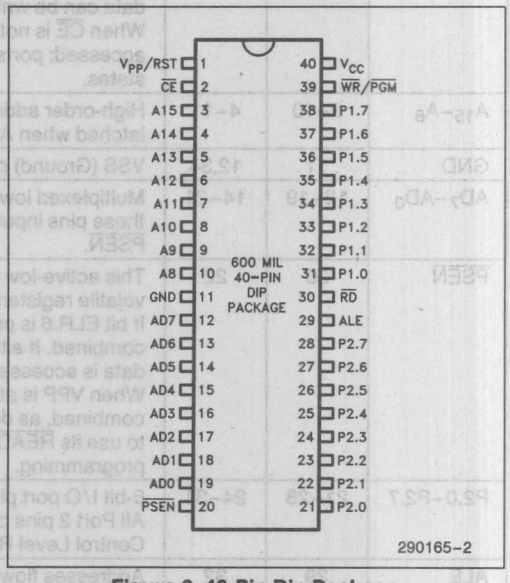


Figure 2. 40-Pin Dip Package

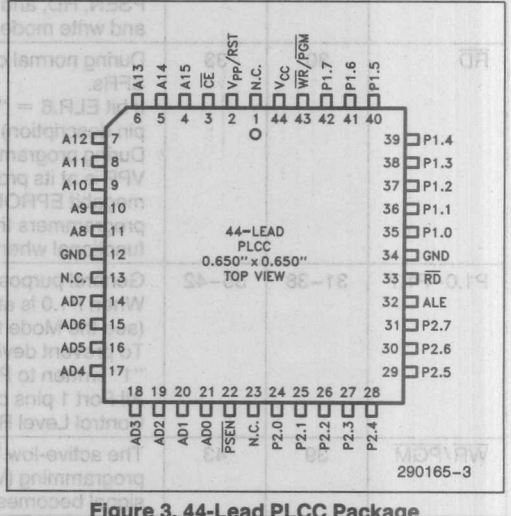


Figure 3. 44-Lead PLCC Package

PIN DESCRIPTIONS

Symbol	Pin Number		Function
	DIP	PLCC	
VPP/RST	1	2	In operating mode, VPP/RST is V_{IL} or V_{IH} and serves as the reset input. RST is user programmable as active-high or active-low via the Control Level Register (CLR.7). When RST is asserted, ports are set to inputs in non-CMOS mode or 1's in CMOS mode. With RST asserted, port-writes have no affect; port-latch-reads return "1s". VPP is the programming supply-voltage input.
$\overline{CE}$	2	3	$\overline{CE}$, the master device enable input, is active-low. When asserted, data can be written and read to/from the device. When $\overline{CE}$ is not asserted, the memory is in standby and cannot be accessed; ports cannot be accessed but maintain their current active states.
A_{15}–A_8	3–10	4–11	High-order addresses flow into the device when ALE = V_{IH} and are latched when ALE = V_{IL}.
GND	11	12,34	VSS (Ground) pins.
AD_7–AD_0	12–19	14–21	Multiplexed low-order address/data. After ALE latches addresses, these pins input or output data depending on $\overline{RD}$, $\overline{WR}/\overline{PGM}$, and $\overline{PSEN}$.
$\overline{PSEN}$	20	22	This active-low pin is the Program Store ENable. EPROM or non-volatile registers are read if this pin is asserted. If bit ELR.6 is programmed ("0"), $\overline{PSEN}$ and $\overline{RD}$ are internally combined. If either or both of these signals is V_{IL}, EPROM or SFR data is accessed depending on the address. When VPP is at its programming voltage, $\overline{PSEN}$ and $\overline{RD}$ are internally combined, as described above. This allows a resident microcontroller to use its $\overline{READ}$ signal to verify programmed data during in-system programming.
P2.0–P2.7	21–28	24–31	8-bit I/O port pins with Quasi-bi-directional (internal pull-up) outputs. All Port 2 pins can be configured as CMOS outputs by programming Control Level Register bit CLR.5.
ALE	29	32	Addresses flow through the latches to address decoders when ALE = V_{IH}. ALE's falling edge latches all addresses independent of $\overline{CE}$. $\overline{PSEN}$, $\overline{RD}$, and $\overline{WR}/\overline{PGM}$ are non-functional when ALE is V_{IH}. Read and write modes are possible only when ALE is V_{IL}.
$\overline{RD}$	30	33	During normal operation, $\overline{RD}$ is used to read information from the SFRs. If bit ELR.6 = "0", $\overline{RD}$ and $\overline{PSEN}$ are internally combined (see $\overline{PSEN}$ pin description). During programming, $\overline{RD}$ and $\overline{PSEN}$ are internally combined when VPP is at its programming voltage. This pin's location is the same as a megabit EPROM's GND pin. For compatibility with PROM programmers that force this pin to ground, $\overline{RD}$ becomes non-functional when P1.0 is at V_H.
P1.0–P1.7	31–38	35–42	General purpose 8-bit open-drain I/O port pins. When P1.0 is at V_H (12V) the Configuration Plane can be accessed (see the Mode table) and $\overline{RD}$ is internally disabled. To prevent device damage, Port 1 must be reset, by RST, or have a "1" written to P1.0 before V_H is applied to P1.0. All Port 1 pins can be configured as CMOS outputs by programming Control Level Register CLR.6.
$\overline{WR}/\overline{PGM}$	39	43	The active-low $\overline{WR}/\overline{PGM}$ is used to write data to the SFRs. During programming (VPP = 12.75V), the SFRs cannot be written, and this signal becomes the program-pulse control input.
VCC	40	44	This pin is the supply voltage input.

EXTENDED TEMPERATURE (EXPRESS) uC PERIPHERAL

Intel's EXPRESS microcontroller and application-specific peripheral families receive additional processing to enhance product characteristics. EXPRESS processing is available for several microcontrollers, EPROMs, and peripheral products allowing the appropriate device to match custom system applications. EXPRESS devices are available with 168 ±8 hour, 125°C dynamic burn-in using Intel's standard bias configuration. This process meets or exceeds most industry burn-in specifications. The standard EXPRESS operating temperature range is 0°C to +70°C. EXPRESS extended operating temperature range (−40°C to +85°C) and automotive temperature range (−40°C to +125°C) products are also available. Like all Intel products, the EXPRESS family is inspected to 0.1% electrical AQL. This allows reduction or elimination of incoming testing.

AUTOMOTIVE AND EXPRESS PRODUCT FAMILY

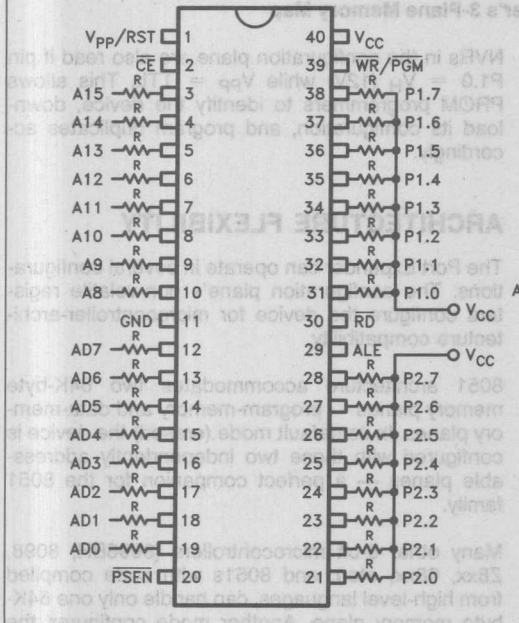
PRODUCT DEFINITIONS

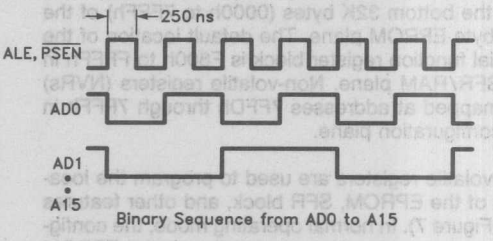
Type	Operating Temperature (°C)	Burn-in 125°C (hr)
Q	0°C to +70°C	168 ±8
T	−40°C to +85°C	NONE
L	−40°C to +85°C	168 ±8

AUTOMOTIVE AND EXPRESS OPTIONS

Speed Versions	Packaging Options	
	CERDIP	PLCC
	Contact your local Intel Sales Office for EXPRESS product availability	

Burn-In Bias and Timing Diagrams





Binary Sequence from AD0 to A15

This cycle also includes port read/write accesses.

V_{CC} = +5V, V_{PP}/RST = +5V, $\overline{CE}$ = GND, $\overline{RD}$ = +5V, $\overline{WR}$/$\overline{PGM}$ = +5V, R = 10K

290165-20

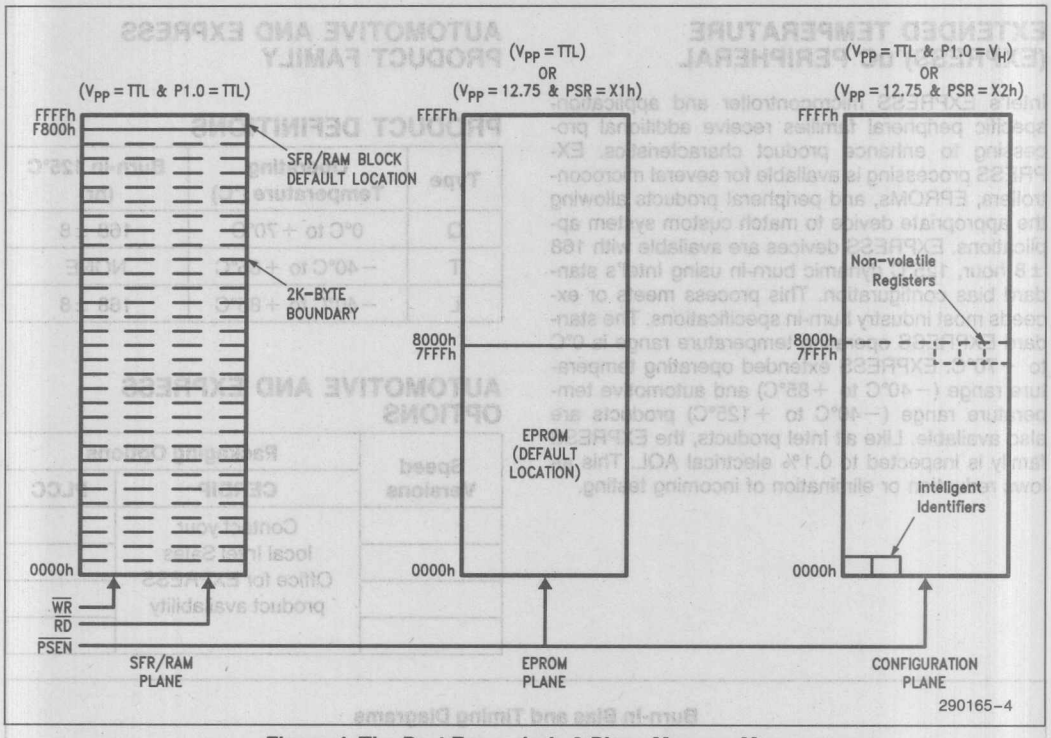


Figure 4. The Port Expander's 3-Plane Memory Map

MEMORY MAP

The Port Expander contains three memory planes — EPROM plane, special function register/RAM (or SFR/RAM) plane, and the configuration plane. Figure 4 shows the three memory planes.

The EPROM's default location (in an erased device) is at the bottom 32K bytes (0000h to 7FFFh) of the 64K-byte EPROM plane. The default location of the special function register block is F800h to FFFFh in the SFR/RAM plane. Non-volatile registers (NVRs) are mapped at addresses 7FFDh through 7FFFh in the configuration plane.

Non-volatile registers are used to program the locations of the EPROM, SFR block, and other features (see Figure 7). In normal operating mode, the configuration plane cannot be accessed; only the EPROM and SFR/RAM planes are available. During programming/verification, the plane select register, PSR, (SFR default location F810h) determines which plane — EPROM or configuration — is accessed. The EPROM array is programmed/verified if PSR contains xxxxxx01b (X1h). The configuration plane is programmed/verified if PSR contains xxxxxx10b (X2h) before V_{PP} is raised to 12.75V.

NVRs in the configuration plane are also read if pin P1.0 = V_H (12V) while V_{PP} = TTL. This allows PROM programmers to identify the device, download its configuration, and program duplicates accordingly.

ARCHITECTURE FLEXIBILITY

The Port Expander can operate in several configurations. The configuration plane's non-volatile registers configure the device for microcontroller-architecture compatibility.

8051 architecture accommodates two 64K-byte memory planes — program-memory and data-memory planes. In its default mode (erased) the device is configured with these two independently addressable planes — a perfect companion for the 8051 family.

Many other 8-bit microcontrollers (8096BH, 8098, Z8xx, 68xx, etc.), and 8051s with code compiled from high-level languages, can handle only one 64K-byte memory plane. Another mode configures the device for single plane operation — again, a perfect 8-bit microcontroller companion device.

Often, more than two external ports and greater than 32K-bytes of external EPROM are required in single-memory-plane applications. Another mode allows two Port Expanders to supply 60K-bytes of EPROM and four 8-bit I/O ports — still leaving 4K-bytes for other read/write devices.

Double-plane Applications

The default configuration has two memory planes; program (EPROM) and data (SFR/RAM). This configuration is consistent with the 8051 architecture. The EPROM plane is read-only and is accessed by $\overline{PSEN}$. The SFR/RAM plane is a read/write plane that is accessed by the $\overline{RD}$ and $\overline{WR}/\overline{PGM}$ inputs. These signals and the sixteen address inputs provide two 64K-byte memory-planes.

Single-plane Applications

Many microcontroller architectures have only one 64K-byte memory plane. One way to configure the device for a single-plane is to simply tie $\overline{PSEN}$ and $\overline{RD}$ together and connect the combined read signal to the system's $\overline{READ}$ line.

8051 machine code compiled from high-level languages often can't deal with separate program- and data-planes. Systems using high-level languages usually form one 64K-byte memory plane by combining $\overline{PSEN}$ and $\overline{RD}$ into a common $\overline{READ}$ signal (by using an AND gate).

The Port Expander provides a better solution. If the EPROM Location Register bit ELR.6 is programmed, $\overline{PSEN}$ and $\overline{RD}$ are combined internally to form a common $\overline{READ}$ signal. Either of these signals can be used to gate data from the EPROM plane and/or SFR/RAM plane to the outputs. In effect, this mode forms a single 64K-byte memory plane. For 8051 high-level-language systems, no external glue is required to "AND" $\overline{PSEN}$ with $\overline{RD}$. The 8051's $\overline{PSEN}$ and $\overline{RD}$ signals can be connected directly to the Port Expander's corresponding inputs. Single-plane, non-8051 microcontroller systems need to route their $\overline{READ}$ line to either, or both, $\overline{PSEN}$ or $\overline{RD}$. If only one input is used, the other must be tied high.

Overlapped Single Plane

Two Port Expanders can fit in a single-memory-plane system by programming the configuration plane's non-volatile registers. To accomplish this, each device must have its SFR block mapped over a portion of its EPROM array. The SFR block can be placed on any 2K-byte boundary by programming the SFRLR. EPROM Location Register bit ELR.7 allows the EPROM to be moved to high memory or to remain in its default low-memory location. Programming ELR.6, the overlap bit, allows the EPROM plane to be mapped over the SFR/RAM plane; this also internally combines $\overline{PSEN}$ and $\overline{RD}$. 2K EPROM bytes located at the SFR block's base-address are disabled and replaced by the 2K-byte SFR block.

Figure 5 shows various memory configurations.

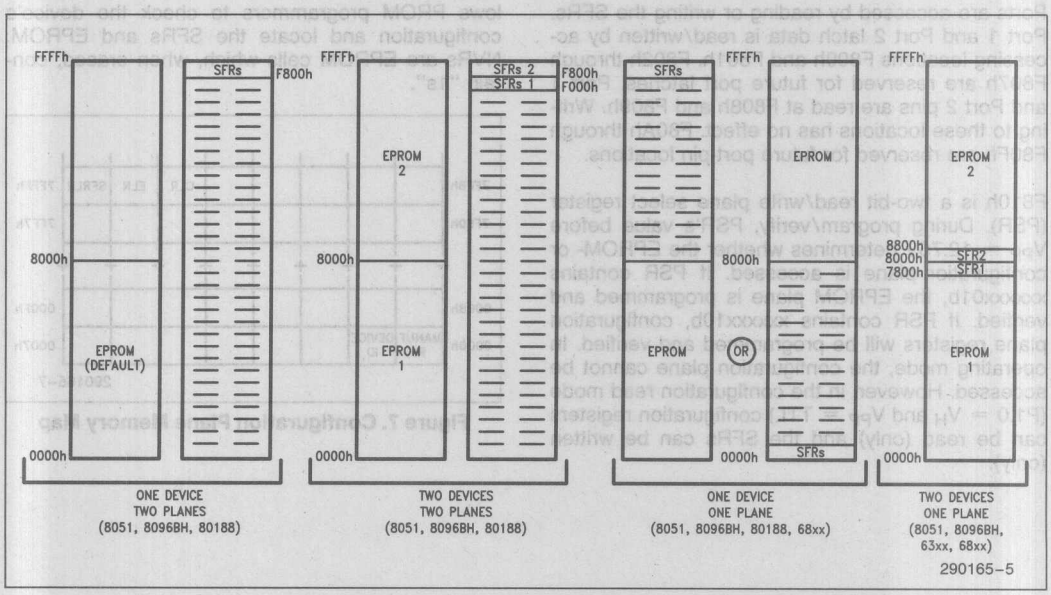


Figure 5. A Few Possible Memory Plane Configurations

EPROM

The Port Expander contains a 32,768 x 8-bit EPROM. When erased, the EPROM is located between EPROM plane addresses 0000h and 7FFFh. This is a common boot-up address range for most microcontrollers, including 8051 and 8096 families. For microcontrollers that reset in high addresses, the EPROM can be relocated to device addresses 8000h through FFFFh via the EPROM Location Register. This also allows systems to use two Port Expanders — one with EPROM at low-memory and the other with EPROM relocated at high-memory.

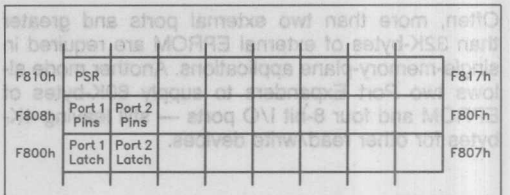
When a valid address is present, $\overline{PSEN}$ controls EPROM access. Asserting $\overline{PSEN}$ during non-valid addresses places device outputs at high impedance.

SPECIAL FUNCTION REGISTERS (SFRs)

SFR addresses described below and in Figure 6 are default in an erased device. A 2K-byte block (default locations F800h through FFFFh) is reserved for ports, plane select register (PSR), and future features. The SFR/RAM-block base address can change depending on the SFRLR's five most-significant bits (Figure 10). Only five SFR/RAM-block locations are defined. Accessing any other addresses in this block places the external bus in a high impedance state allowing external devices to occupy these locations.

Ports are accessed by reading or writing the SFRs. Port 1 and Port 2 latch data is read/written by accessing locations F800h and F801h. F802h through F807h are reserved for future port latches. Port 1 and Port 2 pins are read at F808h and F809h. Writing to these locations has no effect. F80Ah through F80Fh are reserved for future port-pin locations.

F810h is a two-bit read/write plane select register (PSR). During program/verify, PSR's value before V_{PP} = 12.75V determines whether the EPROM- or configuration-plane is accessed. If PSR contains xxxxxx01b, the EPROM plane is programmed and verified. If PSR contains xxxxxx10b, configuration plane registers will be programmed and verified. In operating mode, the configuration plane cannot be accessed. However, in the configuration read mode (P1.0 = V_H and V_{PP} = TTL) configuration registers can be read (only) and the SFRs can be written (only).



NOTE:
SFRs are shown in their default locations.

Figure 6. SFR Memory Map

CONFIGURATION PLANE

Non-Volatile Registers (NVRs)

The configuration plane contains the inteligent Identifier™ and non-volatile registers. This plane is read:

1) if P1.0 = V_H (V_H = 12V ±1V) while V_{PP} = TTL
   or
2) if PSR contains xxxxxx10b while V_{PP} = 12.75V.

Inteligent Identifier codes are at 0000h (manufacturer) and 0001h (device). NVRs are at 7FFDh (CLR), 7FFEh (ELR), and 7FFFh (SFRLR).

NVRs are programmed/verified by writing xxxxxx10b to PSR before V_{PP} = 12.75V; inteligent Identifier bytes are read-only. Figure 7 shows the configuration plane's NVR locations. Condition 1) above allows PROM programmers to check the device's configuration and locate the SFRs and EPROM. NVRs are EPROM cells which, when erased, contain "1s".

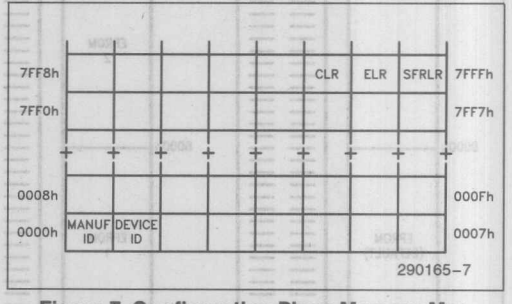


Figure 7. Configuration Plane Memory Map

Control Level Register (CLR)

The Control Level Register, CLR (7FFDh), is used to change the RST pin's active level and port output drive. RST is active-high and CMOS port-drive is disabled in an erased device. If the reset level bit, RSTL (CLR.7), is programmed ("0"), RST is active-low. Port 1 and/or Port 2 outputs will be CMOS if P1C (CLR.6) and/or P2C (CLR.5) are programmed.

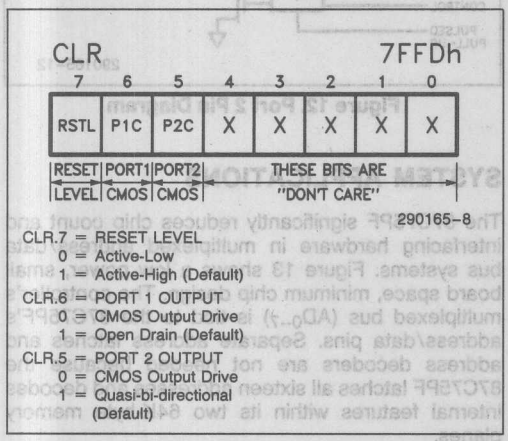


Figure 8. Control Level Register (CLR)

EPROM Location Register (ELR)

The EPROM Location Register (ELR) is at 7FFEh. The EPROM location bit, EL (ELR.7), places the EPROM at either top or bottom EPROM-plane addresses. When erased, the EPROM array is at 0000h–7FFFh in the 64K-byte address space. Programming ELR.7 = "0" places the EPROM at 8000h–FFFFh.

When erased, the overlap option is disabled. EPROM and SFR/RAM blocks are in default locations. $\overline{PSEN}$ accesses EPROM- and $\overline{RD}$ accesses the SFR-data.

If the OVERLAP bit, OVLP (ELR.6), is programmed, EPROM and SFR/RAM planes overlap. $\overline{PSEN}$ and $\overline{RD}$ are internally combined. If either is V_{IL}, EPROM or SFR data is accessed depending on the address.

If the SFR/RAM block's 2K-byte boundary overlaps the EPROM array and ELR.6 = 0, the SFR/RAM block replaces 2K EPROM bytes. Accessing non-defined bytes in the 2K-byte space places the external bus in a high-Z state. By programming ELR.6, one-memory-plane microcontrollers (8096, 80188, 68xx) and 8051s with high-level-language-compiled code can use two 87C75PFs.

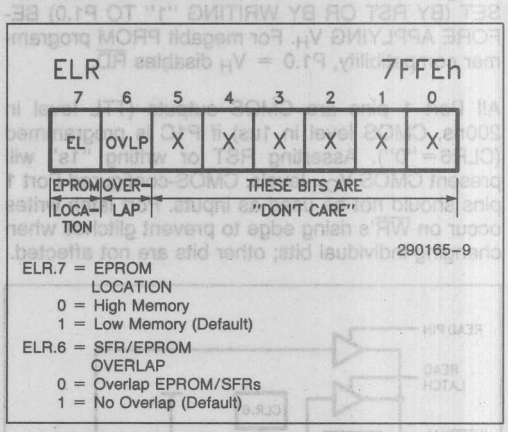


Figure 9. EPROM Location Register (ELR)

SFR Location Register (SFRLR)

The SFRLR (7FFFh) determines the SFR/RAM block's five most-significant base-address bits; SFRLR.7 = A15, SFRLR.6 = A14, SFRLR.5 = A13, SFRLR.4 = A12, and SFRLR.3 = A11. Programming this register places the SFR/RAM block on any 2K-byte boundary. For example, the SFRs are placed at 2800h by programming 00101xxxb.

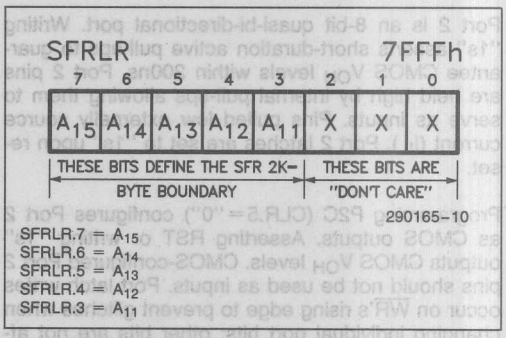


Figure 10. SFR Location Register (SFRLR)

Port 1

Port 1 has 8 open-drain, bi-directional pins. Pins float if "1s" are written to latches or RST is asserted. They can then serve as high impedance inputs.

P1.0 receives high voltage (V_H = 12V) during the inteligent Identifier/NVR Mode. P1.0 MUST BE RESET (BY RST OR BY WRITING "1" TO P1.0) BEFORE APPLYING V_H. For megabit PROM programmer compatibility, P1.0 = V_H disables $\overline{RD}$.

All Port 1 pins are CMOS outputs (TTL level in 200ns, CMOS level in 1us) if P1C is programmed (CLR6="0"). Asserting RST or writing "1s" will present CMOS V_{OH} levels. CMOS-configured Port 1 pins should not be used as inputs. Port latch writes occur on $\overline{WR}$'s rising edge to prevent glitches when changing individual bits; other bits are not affected.

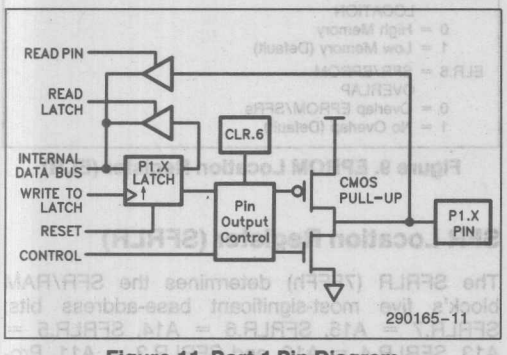


Figure 11. Port 1 Pin Diagram

Port 2

Port 2 is an 8-bit quasi-bi-directional port. Writing "1s" asserts short-duration active pull-ups to guarantee CMOS V_{OH} levels within 200ns. Port 2 pins are held high by internal pull-ups allowing them to serve as inputs. Pins pulled low externally source current (I_{IL}). Port 2 latches are set to "1s" upon reset.

Programming P2C (CLR.5="0") configures Port 2 as CMOS outputs. Asserting RST or writing "1s" outputs CMOS V_{OH} levels. CMOS-configured Port 2 pins should not be used as inputs. Port latch writes occur on $\overline{WR}$'s rising edge to prevent glitches when changing individual port bits; other bits are not affected.

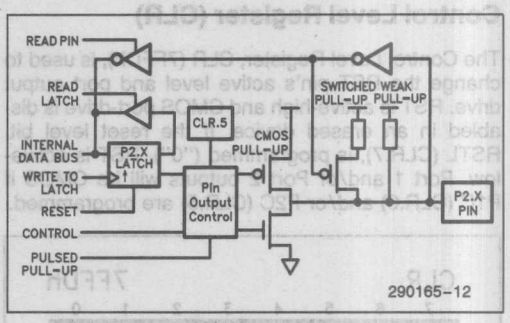


Figure 12. Port 2 Pin Diagram

SYSTEM APPLICATIONS

The 87C75PF significantly reduces chip count and interfacing hardware in multiplexed address/data bus systems. Figure 13 shows a low power, small board space, minimum chip design. The controller's multiplexed bus (AD_{0-7}) is tied to the 87C75PF's address/data pins. Separate address latches and address decoders are not needed because the 87C75PF latches all sixteen addresses and decodes internal features within its two 64K-byte memory planes.

ALE controls the 87C75PF's internal address latches. A V_{IH} to V_{IL} transition latches the present address. $\overline{PSEN}$, $\overline{RD}$, and $\overline{WR}$ control data-flow between the controller and 87C75PF. 8051, 8096, and 80188 families benefit from the 87C75PF's "no-glue" interface.

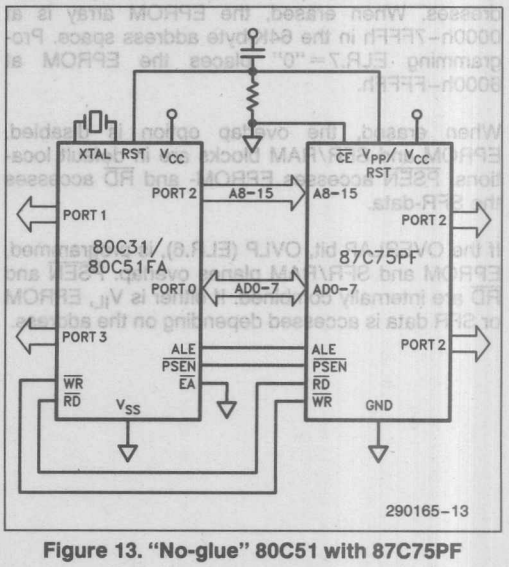


Figure 13. "No-glue" 80C51 with 87C75PF

Table 1. MODE SELECTION 87C75PF (default configuration shown)

MODE	$\overline{CE}$	$\overline{PSEN}$	$\overline{RD}$	$\overline{WR}/\overline{PGM}$	ALE(6)	V_{PP}/RST(4)	V_{CC}	P1.0(2)	AD_{0-7}
Reset	X(1)	X	X	X	X	V_{IH}	5V	X(10)	X
Read EPROM(12)	V_{IL}	V_{IL}	V_{IH}	V_{IH}	V_{IL}	X	5V	X	D Out
Read SFR(12)	V_{IL}	V_{IH}	V_{IL}	V_{IH}	V_{IL}	X	5V	X	D Out
Single Plane Read(8)	V_{IL}	V_{IL} or V_{IL}		V_{IH}	V_{IL}	X	5V	X	D Out
Output Disable	V_{IL}	V_{IH}	V_{IH}	X	V_{IL}	X	5V	X	High Z
Write SFR	V_{IL}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}(14)	5V	X or V_H	D In
Write Disable	V_{IL}	X	X	V_{IH}	V_{IL}	X	5V	X	High Z
Read/Write Disable	V_{IL}	X	X	X	V_{IH}	X	5V	X	High Z
Standby	V_{IH}	X	X	X	X	X	5V	X	High Z
Program EPROM/NVR(5)	V_{IL}	V_{IH}	X(11)	V_{IL}	V_{IL}	V_{PP}(3)	V_{CP}(3)	V_H(7)	D In
EPROM/NVR Verify(5)	V_{IL}	V_{IL}	X(11)	V_{IH}	V_{IL}	V_{PP}	V_{CP}	V_H	D Out
Program Inhibit	V_{IH}	X	X	X	X	V_{PP}	V_{CP}	X	High Z
Alternate Program	V_{IL}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{PP}	V_{CP}	X	D In
Alternate Verify(9)	V_{IL}	V_{IL} or V_{IL}		V_{IH}	V_{IL}	V_{PP}	V_{CP}	X	D Out
NVR Config Read(13)	V_{IL}	V_{IL}	X	V_{IH}	V_{IL}	X(7)	5V	V_H	D Out
inteligent(13) Identifier - Manuf	V_{IL}	V_{IL}	X	V_{IH}	V_{IL}	X(7)	5V	V_H	89h
inteligent(13) Identifier - Device	V_{IL}	V_{IL}	X	V_{IH}	V_{IL}	X(7)	5V	V_H	D0h

NOTES:
1. X can be V_{IL} or V_{IH}.
2. $V_H = 12.0V \pm 1V$.
3. $V_{PP} = 12.75V$ and $V_{CP} = 6.25v$ during programming.
4. RST is active-high (erase default shown) or programmable via CLR.7 as active-low.
5. The EPROM array is programmed/verified if PSR = X1h. The NVR array is programmed/verified if PSR = X2h. NVRs and inteligent Identifier can be read when P1.0 = V_H and V_{PP} = TTL.
6. Data cannot be read/written when ALE = V_{IH}. ALE must toggle — V_{IH} to V_{IL} — to latch addresses.
7. Reset must occur via V_{PP}/RST or "1" written to P1.0 before P1.0 = V_H.
8. If ELR.6 = 0, $\overline{PSEN}$ and $\overline{RD}$ are internally combined.
9. If $V_{PP} = 12.75V$, $\overline{PSEN}$ and $\overline{RD}$ are internally combined. If either is V_{IL}, EPROM (PSR = X1h) or NVR (PSR = X2h) data is verified. If P1.0 = V_H, $\overline{RD}$ is non-functional. If V_{PP} = TTL and P1.0 = V_H, only NVRs and inteligent identifier can be read.
10. RST sets port latches to "1s". After reset, P1.0 (="1") is protected when V_H is applied.
11. For programmer compatibility, the 87C75PF's $\overline{RD}$ is disabled when P1.0 = V_H.
12. $\overline{PSEN}$ and $\overline{RD}$ can be asserted simultaneously unless the EPROM and SFRs overlap & ELR.6 = 1.
13. Addresses must be latched during Identifier/NVR reads.
14. RST not asserted.

DEVICE OPERATION

Table 1 lists 87C75PF operating and programming modes. Operating modes require a 5V power supply. Programming modes require 12.75V V_{PP}, 6.25V V_{CC}, and 12.0V Identifier/NVR-read voltages. All input levels are TTL or CMOS except V_{PP}, V_{CP}, and V_H.

OPERATING MODES

Reset

RST is an active-high input in an erased device. Programming CLR.7 ("0") makes RST active-low. Asserting RST for 500ns sets port latches to "1s". RST affects no other writable locations. Before a PROM programmer enters the inteligent Identifier/NVR read mode, RST should be asserted (or "1" written to P1.0) to set P1.0's pin. This protects P1.0 from damage by the 12.0V identifier voltage.

EPROM Read Mode

$\overline{PSEN}$ enables EPROM data onto AD_{0-7} and controls the device's output buffer. This active-low pin functions only when $\overline{CE}$ and ALE are asserted. When an address is latched (ALE = V_{IL}), access time (t_{AVDV}) equals the $\overline{CE}$ to output delay (t_{CLDV}). Outputs display valid data t_{ELDV} after $\overline{PSEN}$'s falling edge, assuming t_{AVDV} and t_{CLDV} times are met.

SFR Read Mode

$\overline{RD}$ enables SFR data onto AD_{0-7} and controls the device's output buffer. This active-low pin functions only when $\overline{CE}$ and ALE are V_{IL}. EPROM read mode timing requirements apply to this mode.

Single Plane Read Mode

This mode allows single-plane microcontrollers and 8051-family controllers with high-level-language-compiled code to use an 87C75PF without "glue" devices. It is possible to assert $\overline{PSEN}$ and $\overline{RD}$ simultaneously. Data bus conflict will not occur if the SFRs are not memory mapped over EPROM array addresses. If SFR and EPROM addresses overlap, bus conflict can be avoided if the EPROM location register's "Overlap" bit (ELR.6) is programmed. Programming this bit also internally combines $\overline{PSEN}$ and $\overline{RD}$. Asserting either (or both) enables EPROM or SFR data, depending on the address, onto AD_{0-7}. See the "Overlapped Single Plane" section for details.

Output Disable Mode

If $\overline{PSEN}$ and $\overline{RD}$ are not asserted, the device's output buffers (AD_{0-7}) are disabled. Data can be written to the 87C75PF or transferred to/from other devices.

SFR Write Mode

$\overline{WR}/\overline{PGM}$ enables data on AD_{0-7} to be written into the SFRs. This active-low pin functions only when $\overline{CE}$ and ALE are V_{IL}. When an address is latched (ALE = V_{IL}) and data has been present for t_{DVWH}, $\overline{WR}$'s rising edge latches data into an SFR. Other A.C. timing parameters must be observed.

Write Disable

SFR data cannot be written when $\overline{WR}/\overline{PGM}$ is high. Low-address and data share common pins, but the device allows new addresses only when ALE is high; data can be written only when ALE is low.

Read/Write Disable

Since the Port Expander uses a multiplexed address/data bus, data can be read or written only if a valid address is latched. To prevent erroneous reads or spurious writes of invalid data, $\overline{PSEN}$, $\overline{RD}$, and $\overline{WR}/\overline{PGM}$ are non-functional when ALE is high; however, new address information can enter the address latches. ALE's falling edge latches the address and enables $\overline{PSEN}$, $\overline{RD}$, and $\overline{WR}/\overline{PGM}$.

Standby Mode

Standby mode substantially reduces V_{CC} current. $\overline{CE}$ = V_{IH} places output buffers in low-power, high impedance mode independent of $\overline{PSEN}$, $\overline{RD}$, or $\overline{WR}$. Two-line output control ($\overline{CE}+\overline{PSEN}$ or $\overline{CE}+\overline{RD}$) provides:

a) minimum memory power dissipation, and

b) assurance that data bus contention will not occur.

To efficiently use two-line control, address decoding circuitry should enable $\overline{CE}$. $\overline{PSEN}$ should be connected to the microcontroller's program-store enable ($\overline{PSEN}$), $\overline{RD}$ to the controller's data-read enable ($\overline{RD}$), and $\overline{WR}/\overline{PGM}$ to its write control ($\overline{WR}$). This assures that only selected memory and peripheral devices have active inputs and outputs while non-selected devices are in low-power standby mode.

PROGRAMMING MODES

EPROM/Configuration (NVR) Programming Mode

Initially and after each erasure, all EPROM and NVR bits are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" are programmed, the data word can contain both "1s" and "0s". Ultraviolet light erasure is the only way to change "0s" to "1s".

The programming mode is entered when V_{PP} is raised to its programming voltage. After latching an address, data is programmed by applying an 8-bit word to data pins AD_{0-7}. Pulsing $\overline{WR}/\overline{PGM}$ to TTL-low while $\overline{CE}$ and ALE are V_{IL} will program data. TTL levels are required for address and data inputs.

To accommodate PROM programmers that force the $\overline{RD}$ pin to ground (DIP pin 30), applying 12V to port pin P1.0 will internally disable the 87C75PF's $\overline{RD}$ input. When V_{PP} is not at its programming voltage the device is in the inteligent Identifier mode. When V_{PP} is raised for programming, the inteligent Identifier mode is disabled.

EPROM and NVR Verify

With V_{PP} and V_{CC} at their programming levels and $\overline{CE}$ asserted, EPROM or configuration data (depending on PSR's contents) can be verified. To simplify on-board and in-system programming, $\overline{PSEN}$ and $\overline{RD}$ are internally combined when V_{PP} is at its programming level. Either signal can be used to verify programmed data (if P1.0 is not V_H).

For compatibility with PROM programmers equipped for word-wide megabit EPROMs, DIP-pin 30 — the 87C75PF's $\overline{RD}$ pin — is internally disabled when P1.0 is V_H.

Program Inhibit

The Program Inhibit mode allows parallel programming and verification of multiple devices with different data. With V_{PP} at its programming voltage, a $\overline{WE}/\overline{PGM}$ pulse programs any device that has $\overline{CE}$ asserted. Programming is inhibited on any device with $\overline{CE}$ not asserted.

Alternate Programming and Verification Modes

For programmers that can apply V_{IH} or V_{CC} to $\overline{RD}$, the EPROM and NVRs can be programmed using a more conventional slow-motion write-mode-type algorithm. 12V need not be applied to P1.0 to disable the $\overline{RD}$ pin during the alternate programming mode. $\overline{PSEN}$ and $\overline{RD}$ are internally combined when V_{PP} is applied, and either signal can be used to enable EPROM or NVR data during program verification. See the Quick-Pulse Programming algorithm flowchart and waveforms at the end of this data sheet.

inteligent Identifier™/NVR Mode

Programming equipment determines the device's manufacturer, type, and configuration (NVR contents) by using the inteligent Identifier/NVR Mode. A programmer can read a master device's identifier and NVRs, select the proper algorithm, and program duplicates accordingly.

The configuration plane is accessed by raising port pin P1.0 to V_H = 12.0V. Before P1.0 is brought to V_H, Port 1 must be reset by asserting the V_{PP}/RST pin or by writing a "1" to P1.0's latch. When ALE latches a valid address and $\overline{PSEN}$ is V_{IL}, identifier/NVR data appears on Address/Data pins AD_{0-7}. For compatibility with programmers that support megabit EPROMs, $\overline{RD}$, which is usually forced to ground, is "don't-care" when P1.0 = V_H. When $\overline{CE}$, ALE, and $\overline{PSEN}$ are V_{IL} and P1.0 = V_H, identifier/NVR data can be read. While in this mode, the SFR/RAM plane cannot be read but can be written. The PSR register can be configured so that either the EPROM or configuration plane is programmed when V_{PP} is raised. This mode's temperature range is 25°C +5°C.

ABSOLUTE MAXIMUM RATINGS*

Read Operating Temperature . . . . . 0°C to +70°C(2)
Case Temperature Under Bias . . −10°C to +80°C(2)
Storage Temperature . . . . . −65°C to +150°C
All Input or Output Voltages . . . . −2.0V to +7.0V(1)
with Respect to Ground
Voltage on Pin P1.0 . . . . . −2.0V to +13.5V(1)
with Respect to Ground
V_{PP} Supply Voltage . . . . . −2.0V to +14.0V(1)
with Respect to Ground
V_{CC} Supply Voltage . . . . . +2.0V to +7.0V(1)
with Respect to Ground

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

READ/WRITE BUS OPERATION

D.C. CHARACTERISTICS TTL and NMOS Inputs; see A.C. Characteristics for V_{CC} versions offered

Symbol	Parameter	Notes	Min	Max	Units	Test Conditions
I_{LI}	Input Load Current (A_{8-15})			1.0	μA	V_{IN} = 0V to V_{CC}
I_{LO}	Output Leakage Current (AD_{0-7})			10	μA	V_{OUT} = 0V to V_{CC}
I_{SB}	V_{CC} Current Standby	6		5	mA	$\overline{CE}$- inactive, ALE = V_{IL}
I_{CC}	V_{CC} Current Active	4		60	mA	$\overline{CE}$-active, ALE = V_{IH} f(Hz) = $1/t_{AVDV}$, I_{OUT} = 0 mA
V_{IL}	Input Low Voltage	1	−0.5	0.8	V	
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage	1	2.4		V	I_{OH} = −400 μA
I_{OS}	Output Short Circuit Current	5		100	mA	

D.C. CHARACTERISTICS CMOS Inputs; V_{CC} = see A.C. Characteristics for versions offered.

Symbol	Parameter	Notes	Min	Max	Units	Test Conditions
I_{LI}	Input Load Current (A_{8-15})			1.0	μA	V_{IN} = 0V to V_{CC}
I_{LO}	Output Leakage Current (AD_{0-7})			10	μA	V_{OUT} = 0V to V_{CC}
I_{SB}	V_{CC} Current Standby	6		5	mA	$\overline{CE}$-inactive, ALE = V_{IL}
I_{CC}	V_{CC} Current Active	3 4		45	mA	$\overline{CE}$-active, ALE = V_{IH} f(Hz) = $1/t_{AVDV}$, I_{OUT} = 0 mA
V_{IL}	Input Low Voltage	1	−0.2	0.8	V	
V_{IH}	Input High Voltage		0.7 V_{CC}	V_{CC} + 0.2V	V	
V_{OL}	Output Low Voltage			0.40	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage	1	V_{CC} − 0.8		V	I_{OH} = −400 μA
I_{OS}	Output Short Circuit Current	5		100	mA	

NOTES:
1. Minimum DC input voltage is −0.5V during transitions. Inputs may undershoot to −2.0V for periods less than 20 ns. Maximum output-pin DC voltage is V_{CC} + 0.5V; overshoot may be V_{CC} + 2.0V for periods less than 20 ns.
2. This specification defines commercial-product operating temperatures. EXPRESS and Automotive versions are available as noted.
3. $\overline{CE}$ is V_{CC} ± 0.2V (87C75PF inactive) or ± 0.2V (87C75PF active). Other inputs can have any value within specification.
4. Maximum current value with outputs unloaded.
5. One output shorted for no more than one second. I_{OS} is sampled but not 100% tested.
6. Port latches set to "1s"; outputs unloaded.

PORTS/RESET

D.C. CHARACTERISTICS V_{CC} = see A.C. Characteristics for versions offered.

Symbol	Parameter		Notes	Min	Max	Units	Test Conditions
I_{LI}	Input Leakage Current Port 1 Open Drain		1		10	μA	0.4V ≤ V_{IN} ≤ V_{CC}
I_{IL}	Logic 0 current Port 2 (Quasi-bi-directional)		2		−50	μA	P2.x latch = "1", V_{IN} = 0V
V_{IL}	Input Low Voltage	Ports 1&2		−0.5	0.2 V_{CC} − 0.1	V	
		RST Input	3	−0.5	0.2 V_{CC} − 0.1	V	
V_{IH}	Input High Voltage	Ports 1&2		0.2 V_{CC} + .9	V_{CC} + 0.5	V	
		RST Input	3	0.7 V_{CC}	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage				0.40	V	I_{OL} = 3.2 mA
V_{OH}	Output High Voltage	CMOS mode		2.4		V	I_{OH} = −400 μA
		Ports 1&2		0.9 V_{CC}		V	I_{OH} = −40 μA
		Quasi-bi-dir Port 2		2.4		V	I_{OH} = −60 μA
				0.9 V_{CC}		V	I_{OH} = −10 μA

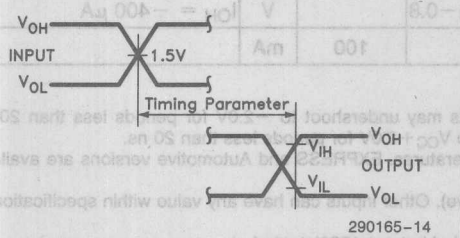
NOTES:
1. Input Leakage current does not apply to Port 2.
2. This specification assumes that Port 2 pins are internally driven to "1" but are externally pulled low
3. RST has hysteresis. V_{IL} is valid at or below 0.2 V_{CC} − 0.1V. V_{IH} is valid at or above 0.7 V_{CC}.

CAPACITANCE[1] T_A = 25°C, f = 1.0MHz

Symbol	Parameter	Max	Units	Conditions
C_{IN}	Address/Control Capacitance	6	pF	V_{IN} = 0V
C_{OUT}	Output Capacitance	12	pF	V_{OUT} = 0V
C_{VPP}	RST/VPP Capacitance	25	pF	V_{IN} = 0V
$C_{I/O}$	Port Pin Capacitance	10	pF	V_{OUT} = 0V

NOTE:
1. Sampled. Not 100% tested.

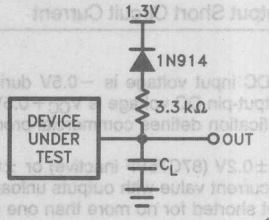
A.C. INPUT/OUTPUT REFERENCE WAVEFORMS



290165-14

A.C. test inputs are driven at V_{OH} (2.4 V_{TTL}) for a logic "1" and V_{OL} (0.45 V_{TTL}) for a logic "0".
Input timing begins at 1.5V.
Output timing ends at V_{IH} (2.0 V_{TTL}) and V_{IL} (0.8 V_{TTL}).
Input Rise and Fall Times (10% to 90%) ≤ 10 ns.

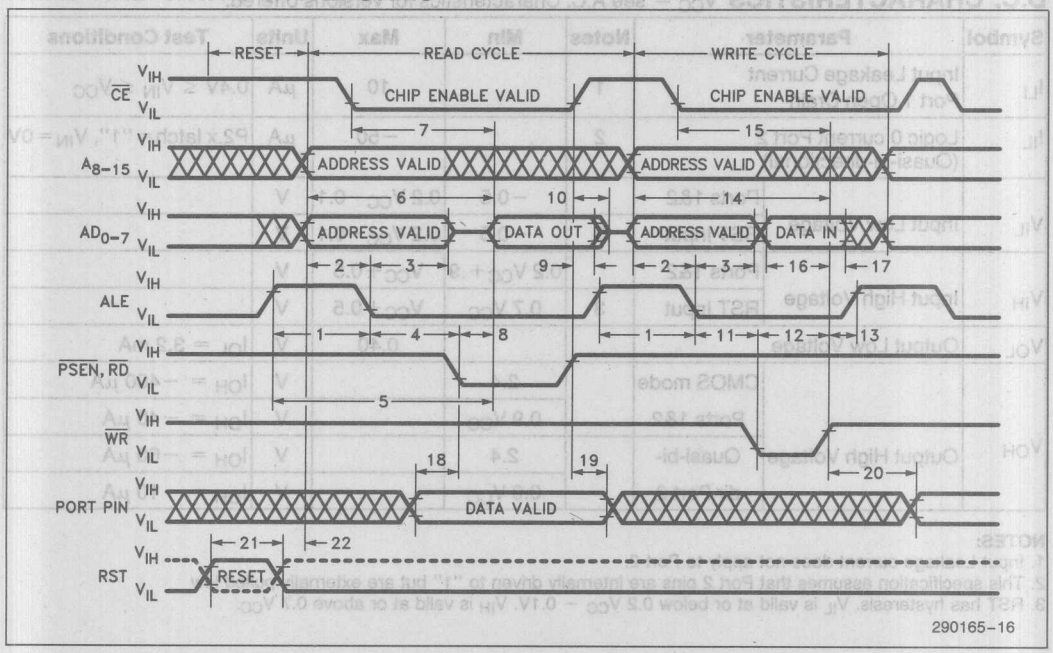
A.C. TESTING LOAD CIRCUIT



290165-15

C_L = 100 pF
C_L Includes Jig Capacitance

READ AND WRITE WAVEFORMS



290165-16

EXPLANATION OF A.C. SYMBOLS

Each timing symbol has five characters. The first is always a "t" (for time). Second and fourth characters represent signal names. Third and fifth represent the signal's logical state. The following list shows character representations.

A: Address
C: Chip Enable or VCC Supply Voltage
D: Data (or instruction)
E: $\overline{PSEN}$ or $\overline{RD}$ Enable
G: $\overline{PGM}$ (Program Strobe)
H: Logic High level
L: ALE or Latch Enable
P: VPP Programming Voltage
Q: Port Output
S: RST (Reset Pin)
T: Time
V: Valid
W: Write Enable
X: No longer a valid "driven" logic level
Z: Float or High-Z level

For example,
t_{AVLL} = Time from Address Valid to ALE Low
t_{LLEL} = Time form ALE Low to Enable ($\overline{PSEN}$ or $\overline{RD}$) Low

A.C. CHARACTERISTICS: READ & WRITE $0°C \leq T_A \leq +70°C$

Parameter		Versions	$V_{CC} \pm 5\%$	87C75PF-200V05		87C75PF-250V05		Unit
No	Symbol	Characteristic	Notes	Min	Max	Min	Max	
1	t_{LHLL}	ALE Pulse Width		50		50		ns
2	t_{AVLL}	Address Valid to ALE Low		7		15		ns
3	t_{LLAX}	Address Hold after ALE Low		20		30		ns
4	t_{LLEL}	ALE Low to $\overline{PSEN}$ or $\overline{RD}$ Low		20		30		ns
5	t_{LHDV}	ALE High to Valid Data			235		305	ns
6	t_{AVDV}	Address Valid to Data Valid	3		200		250	ns
7	t_{CLDV}	$\overline{CE}$ Active to Data Valid	1,3		200		250	ns
8	t_{ELDV}	$\overline{PSEN}$ or $\overline{RD}$ Low to Data Valid	2,3		75		100	ns
9	t_{EHDX}	$\overline{PSEN}$, $\overline{RD}$, $\overline{CE}$, or Address Invalid — Whichever is first — to Data Invalid		0		0		ns
10	t_{EHDZ}	$\overline{PSEN}$ or $\overline{RD}$ High to Data High-Z	4		35		45	ns
11	t_{LLWL}	ALE Low to $\overline{WR}$ Low		20		30		ns
12	t_{WLWH}	$\overline{WR}$ Pulse Width		60		80		ns
13	t_{WHLH}	$\overline{WR}$ High to ALE High		20		30		ns
14	t_{AVWH}	Address Valid to $\overline{WR}$ High		200		250		ns
15	t_{CVWH}	$\overline{CE}$ Active to $\overline{WR}$ High		200		250		ns
16	t_{DVWH}	Data Valid to $\overline{WR}$ High		60		80		ns
17	t_{WHDX}	$\overline{WR}$ High to Data Invalid		10		20		ns
18	t_{QVEL}	Port Input Valid to $\overline{RD}$ Low		15		25		ns
19	t_{EHQX}	Data Hold after $\overline{RD}$ High		0		0		ns
20	t_{WHQV}	$\overline{WR}$ High to Port Output Valid			225		250	ns
21	t_{SVSX}	RST Pulse Width		500		500		ns
22	t_{SXAV}	RST Inactive to Address Valid		0		0		ns

NOTES:
1. t_{CLDV} is 1 μs during inteligent Identifier/NVR Mode.
2. t_{ELDV} is 750 ns during inteligent Identifier/NVR Mode.
3. Output load is 100 pF for t_{AVDV}, t_{CLDV}, and t_{ELDV}.
4. Output Load is 5 pF for t_{EHDZ}, which is measured at high-Z $\pm$500 mV.

PROGRAMMING

Caution: Exceeding 14V on V_{PP} will permanently damage the device.

Program and Data Planes

During programming (V_{PP} = 12.75V), the SFR/RAM plane is not available, only the EPROM and configuration planes (depending on PSR's value) can be accessed. The SFR/RAM plane is accessed only when V_{PP} is V_{IL} or V_{IH}.

Programming the EPROM Plane

The EPROM array is programmed if the plane select register (PSR) contains xxxxxx01b (X1h) when V_{PP} is raised to 12.75V. In an erased device, the EPROM array occupies addresses 0000h through 7FFFh and is programmed at these locations. After programming, the array can be relocated to addresses 8000h–FFFFh by programming the EPROM location register bit ELR.7 (EL) in the configuration plane. Alternately, the EPROM array can be relocated first via ELR.7 and programmed at addresses 8000h–FFFFh.

Programming the Configuration Plane

The configuration plane contains five information bytes. Addresses 0000h and 0001h contain read-only inteligent identifiers. The control level register, EPROM location register, and SFR location register are at 7FFDh, 7FFEh, and 7FFFh. These latter three non-volatile registers (NVRs) are made of EPROM cells. EPROM registers allow the device to be configured, or erased and reconfigured, for various microcontroller architectures.

These registers are programmed if the plane select register (PSR) contains xxxxxx10b (X2h) when V_{PP} is raised to 12.75V. Once this plane is entered, it is programmed and verified just like the EPROM plane.

ERASURE CHARACTERISTICS (FOR CERAMIC, WINDOWED EPROMS)

Exposure to light of wavelength shorter than 4000 Angstroms (Å) begins erasure. Sunlight and some fluorescent lamps have wavelengths in the 3000-4000Å range. Constant exposure to room-level fluorescent light can erase an EPROM in about 3 years (about 1 week for direct sunlight). Opaque labels over the window will prevent unintentional erasure under these lighting conditions.

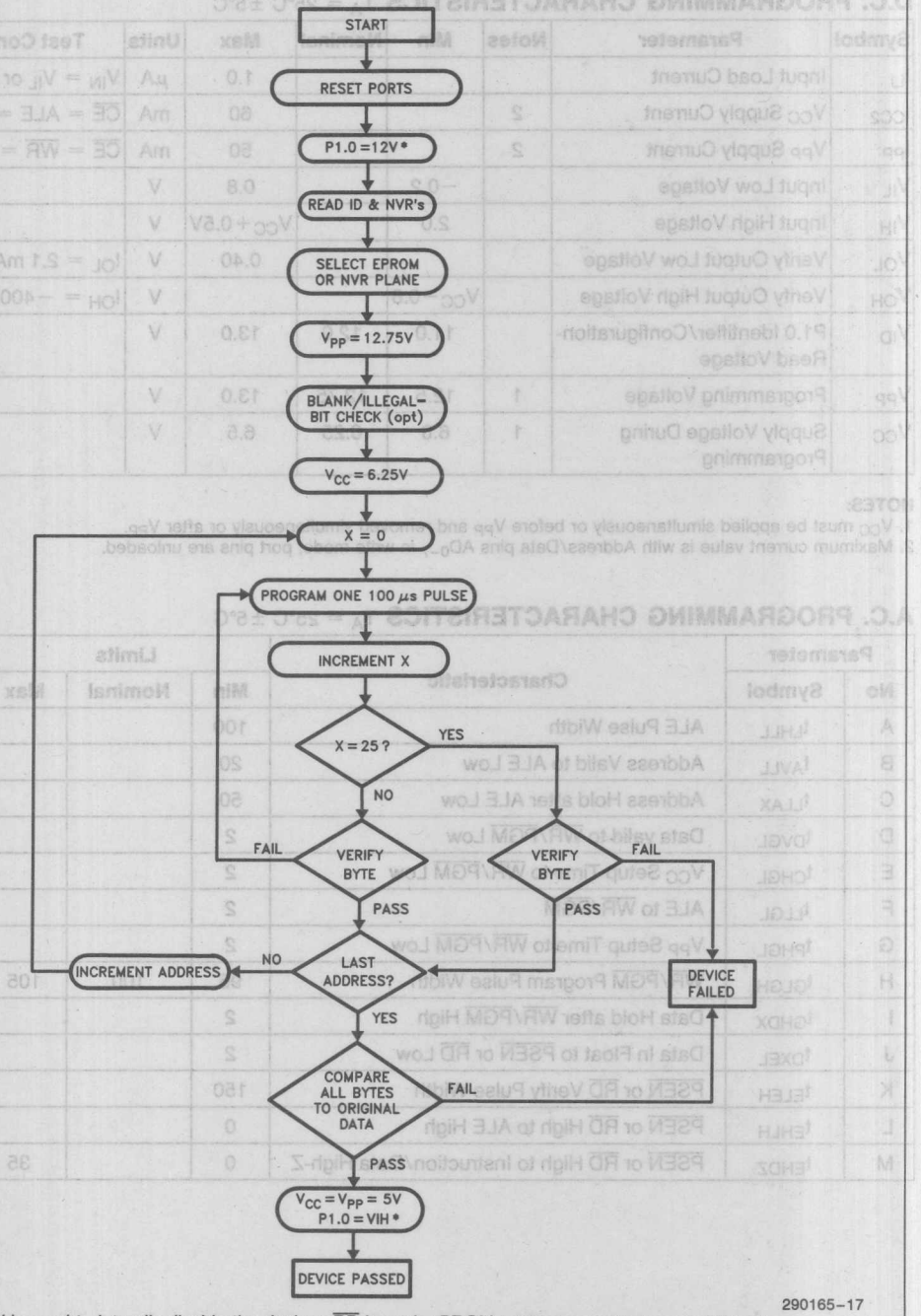
The recommended erasure procedure is exposure to 2537Å ultraviolet light. The minimum integrated Erasure time using a 12000 uW/cm^2 ultraviolet lamp is approximately 15 to 20 minutes. The EPROM should be placed about 1 inch from the lamp. The maximum integrated dose is 7258 Wsec/cm^2 (1 week - 12000 uW/cm^2). High intensity UV light exposure for longer periods can cause permanent damage.

QUICK-PULSE PROGRAMMING™ ALGORITHM

The Quick-Pulse Programming algorithm programs Intel's 87C75PF Port Expander. Developed to substantially reduce production programming throughput time, this algorithm allows optimized programming equipment to program an 87C75PF in under four seconds. Actual programming time depends on the PROM programmer used.

The Quick-Pulse Programming algorithm uses a 100 microsecond initial-pulse followed by a byte verification to determine when the addressed byte is correctly programmed. The algorithm terminates if 25 100us pulses fail to program a byte. Figure 14 shows the 87C75PF Quick-Pulse Programming algorithm flowchart.

The entire program-pulse/byte-verify and final verify sequence is performed with V_{CC} = 6.25V and V_{PP} = 12.75V. When programming is complete, all bytes should be compared to the original data with V_{CC} = 5.0V.



*P1.0 = 12V is used to intrnally disable the devices $\overline{RD}$ input for PROM programmers that ground this pin.

Figure 14. 87C75PF Quick-Pulse Programming™ Algorithm

D.C. PROGRAMMING CHARACTERISTICS $T_A = 25°C \pm 5°C$

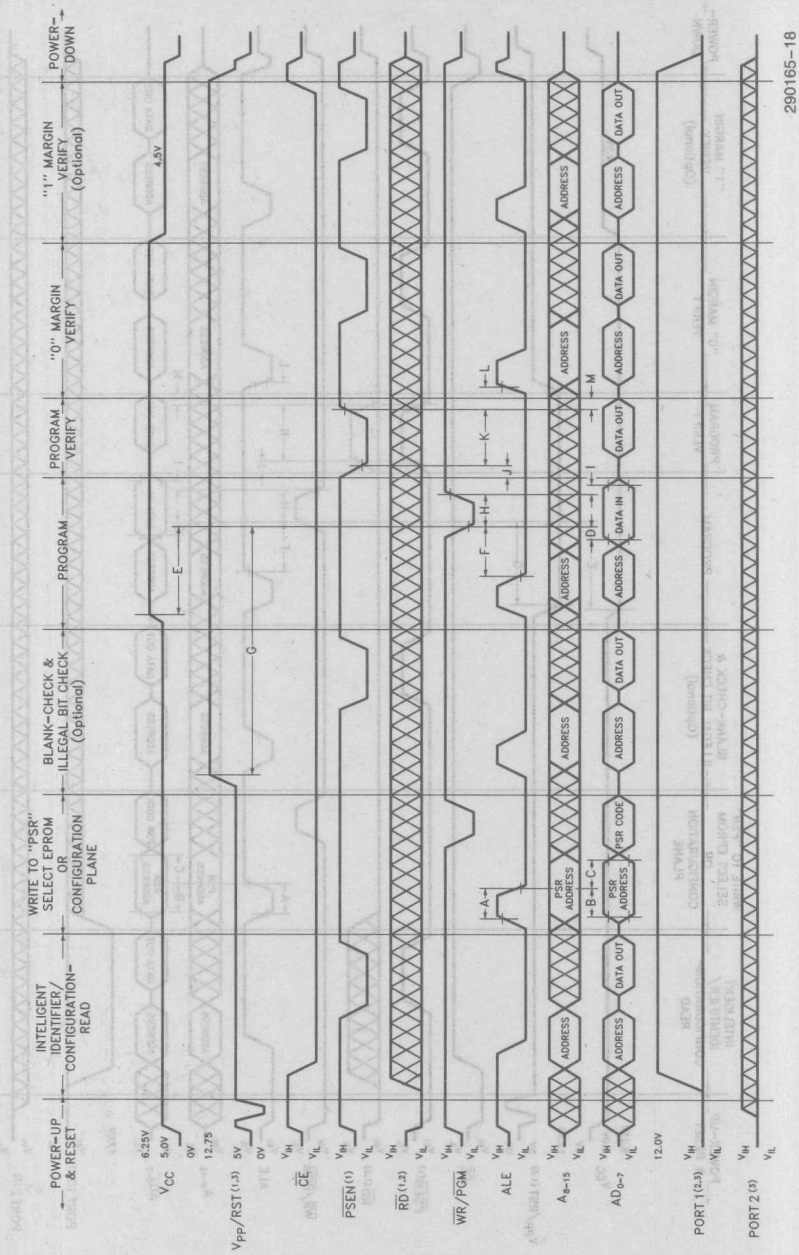
Symbol	Parameter	Notes	Min	Nominal	Max	Units	Test Conditions
I_{LI}	Input Load Current				1.0	μA	$V_{IN} = V_{IL}$ or V_{IH}
I_{CC2}	V_{CC} Supply Current	2			60	mA	$\overline{CE}$ = ALE = V_{IL}
I_{PP}	V_{PP} Supply Current	2			50	mA	$\overline{CE}$ = $\overline{WR}$ = ALE = V_{IL}
V_{IL}	Input Low Voltage		−0.2		0.8	V	
V_{IH}	Input High Voltage		2.0		$V_{CC}+0.5V$	V	
V_{OL}	Verify Output Low Voltage				0.40	V	I_{OL} = 2.1 mA
V_{OH}	Verify Output High Voltage		$V_{CC}-0.8$			V	I_{OH} = −400 μA
V_{ID}	P1.0 Identifier/Configuration-Read Voltage		11.0	12.0	13.0	V	
V_{PP}	Programming Voltage	1	12.5	12.75	13.0	V	
V_{CC}	Supply Voltage During Programming	1	6.0	6.25	6.5	V	

NOTES:
1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. Maximum current value is with Address/Data pins AD_{0-7} in write mode; port pins are unloaded.

A.C. PROGRAMMING CHARACTERISTICS $T_A = 25°C \pm 5°C$

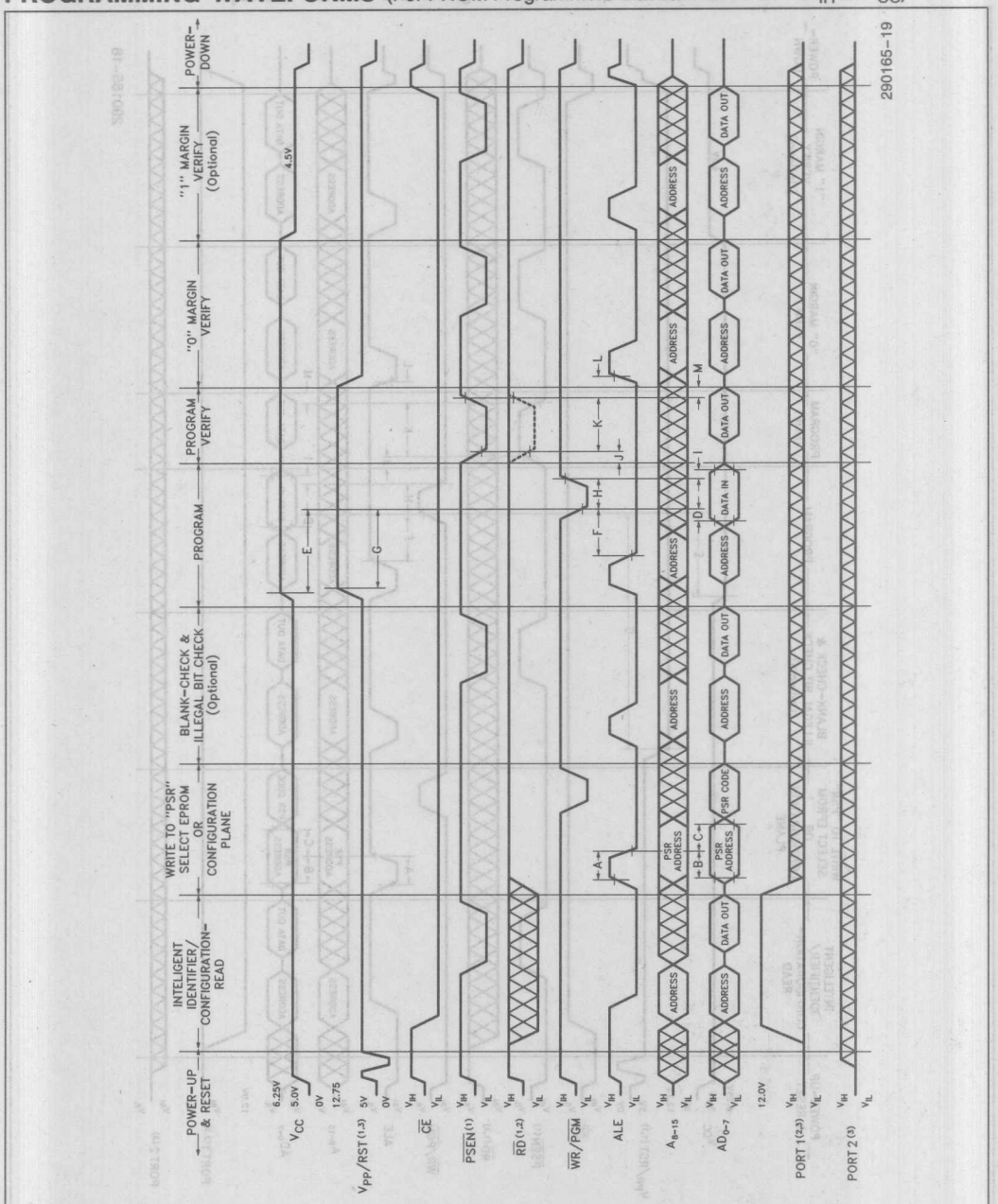
Parameter		Characteristic	Limits			Units
No	Symbol		Min	Nominal	Max	
A	t_{LHLL}	ALE Pulse Width	100			ns
B	t_{AVLL}	Address Valid to ALE Low	20			ns
C	t_{LLAX}	Address Hold after ALE Low	50			ns
D	t_{DVGL}	Data valid to $\overline{WR}/\overline{PGM}$ Low	2			μs
E	t_{CHGL}	V_{CC} Setup Time to $\overline{WR}/\overline{PGM}$ Low	2			μs
F	t_{LLGL}	ALE to $\overline{WR}/\overline{PGM}$	2			μs
G	t_{PHGL}	V_{PP} Setup Time to $\overline{WR}/\overline{PGM}$ Low	2			μs
H	t_{GLGH}	$\overline{WR}/\overline{PGM}$ Program Pulse Width	95	100	105	μs
I	t_{GHDX}	Data Hold after $\overline{WR}/\overline{PGM}$ High	2			μs
J	t_{DXEL}	Data In Float to $\overline{PSEN}$ or $\overline{RD}$ Low	2			μs
K	t_{ELEH}	$\overline{PSEN}$ or $\overline{RD}$ Verify Pulse Width	150			ns
L	t_{EHLH}	$\overline{PSEN}$ or $\overline{RD}$ High to ALE High	0			ns
M	t_{EHDZ}	$\overline{PSEN}$ or $\overline{RD}$ High to Instruction/Data High-Z	0		35	ns

PROGRAMMING WAVEFORMS (For PROM Programmers with $\overline{RD}$ = GND)



NOTES:
1. When V_{PP} = Programming Voltage, either $\overline{PSEN}$ or $\overline{RD}$ will access EPROM (if PSR = x1h) or Configuration (if PSR = x2h) Planes and inteligent Identifier mode is disabled.
2. $\overline{RD}$ is Don't-care when P1.0 = V_H.
3. Port 1 must be RESET or "1" written to P1.0 before V_H is applied to P1.0. All other Port 1 and Port 2 pins should be driven at High-Z or V_{IH} during programming.

PROGRAMMING WAVEFORMS (For PROM Programmers that can have $\overline{RD}$ = V_{IH} or V_{CC})



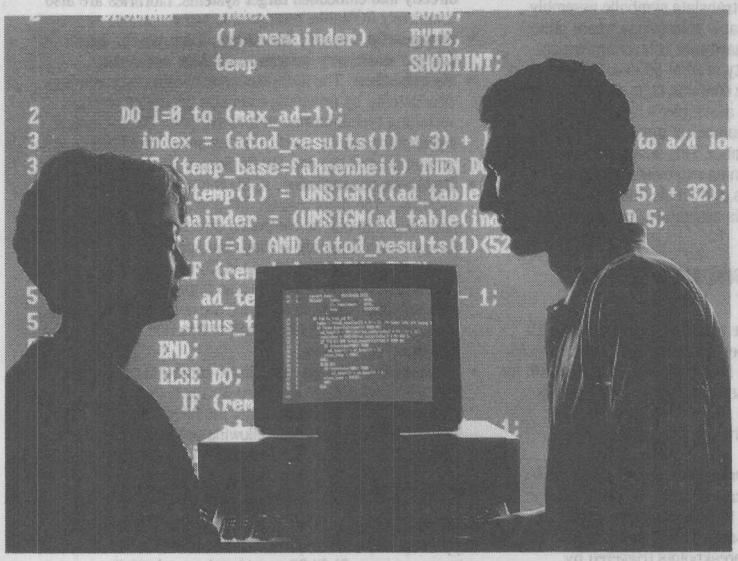
NOTES:
1. When V_{PP} = Programming Voltage, either $\overline{PSEN}$ or $\overline{RD}$ will access EPROM (if PSR = x1h) or Configuraton (if PSR = x2h) Planes and inteligent Identifier mode is disabled.
2. $\overline{RD}$ is Don't-care when P1.0 = V_H.
3. Port 1 must be RESET or "1" written to P1.0 before V_H is applied to P1.0. All other Port 1 and Port 2 pins should be driven at High-Z or V_{IH} during programming.

80186 Development Support Tools

10

10 80186 Development Support Tools

8086/80186 SOFTWARE DEVELOPMENT PACKAGES



COMPLETE SOFTWARE DEVELOPMENT SUPPORT FOR THE 8086/80186 FAMILY OF MICROPROCESSORS

Intel supports application development for the 8086/80186 family of microprocessors (8086, 8088, 80186, 80188 and real mode 80286 and 80386 designs) with a complete set of development languages and utilities. These tools include a macro assembler and compilers for C, PL/M, FORTRAN and Pascal. A linker/relocator program, library manager, numerics support libraries, and object-to-hex utility are also available. Intel software tools generate fast and efficient code. They are designed to give maximum control over the processor. Most importantly, they are designed to get your application up and running in an embedded system fast and with maximum design productivity.

FEATURES

- Macro assembler for speed-critical code
- NEW windowed, interactive source level debugger works with all Intel languages to speed functional debug of code
- ANSI Compatible iC-86 package for structured C programming, with many processor specific extensions
- PL/M compiler for reliable, easy-to-maintain high-level language programs with support for many low-level hardware functions
- FORTRAN for ANSI-compatible programming of numeric intensive applications
- Pascal for developing modular, portable applications that are easy to maintain
- Linker program for linking modules generated by Intel compilers and assemblers into relocatable object modules and direct creation of files that can be executed on the DOS host
- Locator for generating programs with absolute addresses for execution from ROM based systems
- AEDIT Source Code and text editor
- Library manager for creating and maintaining software object module libraries
- Complete numeric support libraries for use with the 8087 or 80C187, including a software emulator for true emulation of the 8087 in applications where the 8087 is not available
- Object-to-hex conversion utility for burning code into (E)PROMS
- Hosted on IBM PC XT/AT* or compatibles running DOS, DEC VAX* or MicroVAX* systems running VMS, and Intel Systems 86/3XX or 286/3XX running iRMX® Operating System

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1988

October, 1988
Order Number: 280808-001

ASM-86 MACRO ASSEMBLER

ASM-86 is the macro assembler for the 8086/80186 family of components. It is used to translate symbolic assembly language source into relocatable object code where utmost speed, small code size and hardware control are critical. Intel's exclusive macro facility in ASM-86 saves development and maintenance time, since common code sequences need only be developed once. The assembler's simplified instruction set reduces the number of mnemonics that the programmer needs to remember. This assembler also saves development time by performing extensive checks on consistent usage of variables and labels. Inconsistencies are detected when the program is assembled, before linking or debugging is started.

NEW FOR 1989: SOURCE LEVEL DEBUGGER

DB-86 is an on-host software execution envrionment with source level debug capabilities for object modules produced by iC-86, ASM-86, PL/M-86, Pascal-86 and FORTRAN-86. Its powerful, source-oriented interface allows users to focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

- **Ease of learning.** Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems.
- **Extensive debug modes.** Watch windows, conditional breakpoints (breakpoints triggered by program conditions), trace points, and fixed and temporary breakpoints can be set and modified as needed.
- **See into your program.** You can browse source and call stack, observe processor registers, output screen, and watch window variables accessed by either the pull down menu or by a single keystroke using function keys.
- **Full debug symbolics for maximum productivity.** The user need not know whether a variable is an unsigned integer, a real, or a structure; the debugger utilizes the wealth of variable typing information available in Intel languages to display program variables in their respective type formats.
- **Support for overlaid programs and the numeric coprocessor.**

iC-86 SOFTWARE PACKAGE

Intel's iC-86 brings the full power of the C programming language to 8086, 8088, 80186 and 80188-based microprocessor systems. It can also be used to develop real mode programs for execution on the 80286 or 80386. iC-86 has been developed specifically for embedded microprocessor-based applications. iC-86 meets the draft proposed ANSI C standard. Key features of the iC-86 compiler include:

- **Highly Optimized.** Four levels of optimization are available. Important optimization features include a jump optimizer and improved register manipulation via register history.
- **ROMable Code and Libraries.** The iC-86 compiler produces ROMable code which can be loaded directly into embedded target systems. Libraries are also completely ROMable, retargetable and reentrant.
- **Supports Small, Medium, Compact, and Large memory segmentation models.**
- **Symbolics.** The iC-86 compiler boosts programming productivity by providing extensive debug information, including type information and symbols. The symbolics information can be used to debug using Intel ICE™ emulators and the new DB-86 Source level debugger.
- **Built-in functions.** iC-86 is loaded with built-in functions. The flags register, I/O ports, interrupts, and numerics chip can be controlled directly, without the need for assembly language coding. You spend more of your productive time programming in C and less with Assembler. Built-in functions also improve compile-time and run-time performance since the compiler generates in-line code instructions instead of function calls to assembly instructions.
- **Standard Language.** iC-86 conforms to the 1988 Draft Proposed ANSI standard for the C language. iC-86 code is fully linkable with other modules written in other Intel 8086/186 languages, allowing programmers to use the optimal language for any task.

PL/M-86 SOFTWARE PACKAGE

PL/M-86 is a high-level programming language designed to support the software requirements of advanced 16-bit microprocessors. PL/M-86 provides the productivity advantages of a high-level language while providing the low-level hardware access features of assembly language. Key features of PL/M-86 include:

- **Structured programming.** PL/M-86 supports modular and structured programming, making programs easier to understand, maintain and debug.
- **Built-in functions.** PL/M-86 includes an extensive list of functions, including TYPE CONVERSION functions, STRING manipulations, and functions for interrogating 8086/186 hardware flags.
- **Interrupt handling.** The INTERRUPT attribute allows you to define interrupt handling procedures. The compiler generates code to save and restore all registers for INTERRUPT procedures.
- **Compiler controls.** Compile-time options increase the flexibility of the PL/M-86 compiler. They include: optimization, conditional compilation, the inclusion of common PL/M source files from disk, cross-reference of symbols, and optional assembly language code in the listing file.
- **Data types.** PL/M-86 supports seven data types, allowing the compiler to perform three different kinds of arithmetic: signed, unsigned and floating point.
- **Language compatibility.** PL/M-86 object modules are compatible with all other object modules generated by Intel 8086/186 languages.

FORTRAN-86 SOFTWARE PACKAGE

FORTRAN-86 meets the ANSI FORTRAN 77 Language Subset Specification and includes almost all of the features of the full standard. This compatibility assures portability of existing FORTRAN programs and shortens the development process, since programmers are immediately productive without retraining.

FORTRAN-86 provides extensive support for numeric processing tasks and applications, with features such as:

- Support for single, double, double extended precision, complex, and double complex floating-point data types
- Support for proposed REALMATH IEEE floating point standard
- Full support for all other data types: integer, logical and character
- Optional hardware (8087 numeric data processor) or software (simulator) floating-point support at link time

PASCAL-86 SOFTWARE PACKAGE

Pascal-86 conforms to the ISO Pascal standard, facilitating application portability, training and maintenance. It has also been enhanced with microcomputer support features such as interrupt handling, direct port I/O and separate compilation.

A well-defined and documented run-time operating system interface allows the user to execute applications under user-designed operating systems as an alternate to the development system environment. Program modules compiled under Pascal-86 are compatible and linkable with modules written in other Intel 8086/186 languages, so developers can implement each module in the language most appropriate for the task at hand.

Pascal-86 object modules contain symbol and type information for program debugging using Intel ICE™ emulators and the DB-86 debugger.

LINK-86 LINKER

Intel's LINK-86 utility is used to combine multiple object modules into a single program and resolve references between independently compiled modules. The resulting linked module can be either a bound load-time-locatable module or simply a relocatable module. A .EXE option allows modules to be generated which can be executed directly on a DOS system.

LINK-86 greatly increases productivity by allowing you to use modular programming. The incremental link capability allows new modules to be easily added to existing software. Because applications can be broken into separate modules, they're easier to design, test and maintain. Standard modules can be reused in different applications, saving software development time.

LOC-86 LOCATOR

The LOC-86 utility changes relocatable 8086/186 object modules into absolute object modules. Its default address assignment algorithm will automatically assign absolute addresses to the object modules prior to loading of the code into the target system. This frees you from concern about the final arrangement of the object code in memory. You still have the power to override the control and specify absolute addresses for various Segments, Classes, and Groups in memory. You may also reserve various parts of memory.

LOC-86 is a powerful tool for embedded development because it simplifies set up of the bootstrap loader and initialization code for execution from ROM based systems. The locator will also optionally generate a print file containing diagnostic information to assist in program debugging.

NUMERICS SUPPORT LIBRARY

The 8087 Support Library (80C187 support in June 1989) greatly facilitates the use of floating-point calculations from programs written in Assembler, PL/M, and C. It adds to these languages many of the functions that are built into applications programming languages, such as Pascal and FORTRAN. A full 8087 software emulator and interface libraries are included for precision floating point calculations without the use of the 8087 component. The decimal conversion library aids the translation between decimal and binary formats. A Common Elementary Function library provides support for transcendental, rounding and other common functions, not directly handled by the numeric processor. An Error Handler Module makes it easy to write interrupt routines that recover from floating-point error conditions.

LIB-86 LIBRARIAN

The Intel LIB-86 utility creates and maintains libraries of software object modules. Standard modules can be placed in a library and linked to your application using the LINK-86 utility.

AEDIT SOURCE CODE AND TEXT EDITOR

AEDIT is a full-screen text editing system designed specifically for software engineers and technical writers. With the facilities for automatic program block indentation, HEX display and input, and full macro support, AEDIT is an essential tool for any programming environment. And with AEDIT, the output file is the pure ASCII text (or HEX code) you input—no special characters or proprietary formats.

Dual file editing means you can create source code and its supporting documents at the same time. Keep your program listing with its errors in the background for easy reference while correcting the source in the foreground. Using the split-screen windowing capability, it is easy to compare two files, or copy text from one to the other. The DOS system-escape command eliminates the need to leave the editor to compile a program, get a directory listing, or execute any other program executable at the DOS system level.

OH-86 OBJECT-TO-HEXADECIMAL CONVERTER

The OH-86 utility converts Intel 8086/186 object modules into standard hexadecimal format, allowing the code to be loaded directly into PROM using industry standard PROM programmers.

SERVICE, SUPPORT AND TRAINING

Intel augments its 8086/186 family development tools with a full array of seminars, classes and workshops. In addition, on-site consulting services, field application engineering expertise, telephone hotline support, and software and hardware maintenance contracts are available to help assure your design success.

ORDERING INFORMATION

D86ASM86NL	ASM-86	Assembler for PC XT or AT system (or compatible) running DOS 3.0 or higher
VVSASM86	ASM-86	Assembler for VAX/VMS
MVVSASM86	ASM-86	Assembler for MicroVAX/ VMS
R86ASM86SU	ASM-86	Assembler for Intel 86/3XX systems running iRMX 86 operating system
R286ASM86EU	ASM-86	Assembler for Intel 286/3XX systems running iRMX II™ operating system
Note:		ASM-86 includes Macro Assembler, Link-86, Loc-86, Lib-86, Cross-Reference utility, OH-86, Numerics Support, and DB-86 Source Level Debugger. (DB-86 available in DOS version only.)
D86C86NL	iC-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher
VVSC86	iC-86	Software Package for VAX/ VMS
MVVSC86	iC-86	Software Package for MicroVAX/VMS
R86C86SU	iC-86	Software Package for Intel System 8086/3XX running iRMX 86 operating system
D86PLM86NL	PL/M-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher
VVSPLM86	PL/M-86	Software Package for VAX/ VMS
MVVSPLM86	PL/M-86	Software Package for MicroVAX/VMS
R86PLM86SU	PL/M-86	Software Package for Intel System 8086/3XX running iRMX 86 operating system
D86FOR86NL	FORTRAN-86	Software Package for PC XT/ AT (or compatible) running PC-DOS 3.0 or higher
VVSFORT86	FORTRAN-86	Software Package for VAX/ VMS 4.3 and later
MVVSFORT86	FORTRAN-86	Software Package for MicroVAX/VMS
R86FOR86SU	FORTRAN-86	Software Package for Intel System 86/3XX running iRMX 86 operating system
D86PAS86NL	PASCAL-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher
VVSPAS86	PASCAL-86	Software Package for VAX/ VMS
MVVPAS86	PASCAL-86	Software Package for MicroVAX/VMS
R86PAS86SU	PASCAL-86	Software Package for Intel System 86/3XX running iRMX 86
D86EDNL		AEDIT Source Code Editor for IBM PC XT/AT running PC DOS 3.0 or higher

ICE is a trademark and iRMX a registered trademark of Intel Corporation.
VAX and VMS are registered trademarks of Digital Equipment Corporation.

VAX*/VMS* RESIDENT 86/88/186/188 SOFTWARE DEVELOPMENT PACKAGES

- **Executes on DEC VAX*/MicroVAX Minicomputer under VMS* Operating System to translate PL/M-86, Fortran-86, iC-86, Pascal-86 and ASM-86 Programs for 8086, 88, 186 and 188 Microprocessors.**
- **Specifically Designed for Embedded Microcomputer Applications**
- **Packages include C-86; FORTRAN-86; Pascal-86; PL/M-86; ASM-86; Link and Relocation Utilities; OH-86 Absolute Object Module to Hexadecimal Format Converter; and Library Manager Program.**
- **Output Linkable with Code Generated on PC-DOS Based Systems**

The VAX/VMS Resident Software Development Packages contain software development tools for the 8086, 88, 186, and 188 microprocessors. The tools allow the user to develop, compile, maintain libraries, and link and locate programs on a VAX running the VMS operating system. The compiler or assembler output is object module compatible with programs translated by the corresponding version of the compiler or assembler on a PC-DOS based system.

Four packages are available:

1. An ASM-86 Assembler Package which includes the Assembler, the Link Utility, the Locate Utility, the absolute object to hexadecimal format conversion utility, the Library Manager Program, and Numerics Support Library.
2. A PL/M-86 Compiler Package which contains the PL/M-86 Compiler and Runtime Support Libraries.
3. A Pascal-86 Compiler Package which contains the Pascal-86 Compiler and Runtime Support Libraries.
4. A iC-86 Compiler Package which contains the iC-86 Compiler and Run-Time Libraries.
5. A FORTRAN-86 Compiler Package which contains the FORTRAN-86 Compiler and Run-Time Libraries.

The VAX/VMS resident development packages and the PC-DOS hosted development packages are built from the same technology base. Therefore, the development packages are very similar.

Version numbers can be used to identify features correspondence. The VAX/VMS resident development packages will have the same features as the PC-DOS hosted product with the same version number.

The object modules produced by the translators contain symbol and type information for program debugging using ICE™ translators and/or the PSCOPE debugger. For final production version, the compiler can remove this extra information and code, to reduce final program size.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

October 1988
Order Number: 210643-005

VAX*-PL/M-86/88/186/188 SOFTWARE PACKAGE

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Standards**
- **Easy-To-Learn Block-Structured Language Encourages Program Modularity**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX*, or a PC-DOS Hosted System**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**

VAX-PL/M-86 is an advanced, structured high-level programming language. The VAX-PL/M-86 compiler was created specifically for embedded software development for the Intel 8086, 88, 186, and 188 microprocessors.

PL/M is a powerful, structured, high-level system implementation language in which program statements can naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The VAX-PL/M-86 compiler efficiently converts free-form PL/M language statements into equivalent machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-PASCAL-86/88/186/188 SOFTWARE PACKAGE

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on either a VAX*, or a PC-DOS Hosted System**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**
- **Supports Numeric Coprocessors**
- **Conforms to IEEE/ANSI and ISO Standards**
- **Extensions for Embedded Microcomputer Applications**
- **Separate Compilation with Type-Checking Enforced between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**
- **Source Input/Object Output Compatible with Pascal-86 Hosted on a PC-DOS or RMX Development Host**

VAX-PASCAL-86 conforms to and implements the ISO Pascal standard, level 0. The language is enhanced to support embedded microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. Other extensions include additional data types not required by the standard and miscellaneous enhancements such as an allowed underscore in names, and an OTHERWISE clause in CASE statements. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The VAX-PASCAL-86 compiler runs on the Digital Equipment Corporation VAX under the VMS Operating System. A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs on the target system as an alternate to the development system environment. Program modules compiled under PASCAL-86 are compatible and linkable with modules written in PL/M-86, and ASM-86. With a complete family of compatible programming languages for the 8086, 88, 186, and 188, code can be developed in the language most appropriate to the task at hand.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX* ASM-86/88/186/188 MACRO ASSEMBLER

- **Executes on VAX*/MicroVAX Minicomputers under The VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to Other Intel 8086/88/186 Object Modules, Generated on either a VAX*, or a PC XT/AT running PC-DOS**
- **Powerful and Flexible Text Macro Facility with Three Macro Listing Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **"Strongly Typed" Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as "STRUCTURES" and "RECORDS"**
- **Detailed and Fully Documented Error Messages**
- **Produces Relocatable and Linkable Object Code**
- **Source Input/Object Output Compatible with ASM-86 hosted on a PC-DOS Hosted System**

VAX-ASM-86 is the "high-level" macro assembler for the 8086/88/186/188 assembly language. VAX-ASM-86 translates symbolic assembly language mnemonics into relocatable object code.

VAX-ASM-86 should be used where maximum code efficiency and hardware control is needed. The assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible machine instructions. VAX-ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

VAX-ASM-86 offers many features normally found only in high-level languages. The ASM-86 assembly language is strongly typed. The assembler performs extensive checks on the usage of variable and labels. The assembler uses the attributes which are derived explicity when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be deteced when the program is assembled, long before it is being debugged on hardware.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LIB-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **VAX-LIB-86 is a Library Manager Program which Allows You to:**
  Create Specifically Formatted Files to Contain Libraries of Object Modules
  Maintain These Libraries by Adding or Deleting Modules
  Print a Listing of the Modules and Public Symbols in a Library File
- **Libraries Can be Used as Input to VAX-LINK-86 Which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program VAX-LIB-86 creates and maintains files containing object modules. The operation of VAX-LIB-86 is controlled by commands to indicate which operation VAX-LIB-86 is to perform. The commands are:

CREATE:	creates an empty library file
ADD:	adds object modules to a library file
DELETE:	deletes modules from a library file
LIST:	lists the module directory of library files
EXIT:	terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

VAX-OH-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Converts an 8086/88/186/188 Absolute Object Module to Symbolic Hexademical Format**
- **Facilitates Preparing a file for Loading by Symbolic Hexadecimal Loader (e.g. iSBC® Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The VAX-OH-86 utility converts an absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from VAX-LOC-86 is in absolute format.

*VAX, VMS are trademarks of Digital Equipment Corporation.

VAX*-LINK-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Combination of Separately Compiled or Assembled 86/88/186/188 Programs into a Relocatable Module, Generated on Either a VAX, or a PC XT/AT running PC-DOS**
- **Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References**
- **Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Automatic Generation of a Summary Map Giving Results of the LINK-86 Process**
- **Abbreviated Control Syntax**
- **Relocatable modules may be Merged into a Single Module Suitable for Inclusion in a Library**
- **Supports "Incremental" Linking**
- **Supports Type Checking of Public and External Symbols**

VAX-LINK-86 combines object modules specified in the VAX-LINK-86 input list into a single output module. VAX-LINK-86 combines segments from the input modules according to the order in which the modules are listed.

VAX-LINK-86 will accept libraries and object modules built from any Intel translator generating 8086 Relocatable Object Modules.

Support for incremental linking is provided since an output module produced by VAX-LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

VAX-LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistent.

VAX-LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the VAX-LINK-86 process and to control the content of the output module.

VAX-LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using VAX-LOC-86 and enter final testing with much of the work accomplished.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LOC-86

- **Executes on the VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and their Addresses**
- **Extensive Capability to Manipulate the Order and Placement of Segments in Memory**
- **Abbreviated Control Syntax**
- **Automatic and Independent Relocation of Independent Relocation of Segments. Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

VAX-LOC-86 converts relative addresses in an input module in iAPX-86/88/186/188 object module format to absolute addresses. VAX-LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

VAX-LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

SPECIFICATIONS

Operating Environment

Required Hardware

VAX* with 9 Track Magnetic Tape Drive, 1600 BPI

MicroVAX with TK-50 tape drive.

Required Software

VMS Operating System. All of the development packages are delivered as unlinked VAX object code which can be linked to VMS as designed for the system where the development package is to be used. VMS command files to perform the link are provided.

MicroVMS

Documentation Package

iAPX-86, 88 Development Software Installation Manual and User's Guide for VAX/VMS, Order number 121950-001

Shipping Media

9 Track Magnetic Tape 1600 bpi (VAX)

TK-50 Cartridge Tape (MicroVAX)

ORDERING INFORMATION

Part Number	Description
VVSASM86	VAX-ASM-86, VAX-LINK-86, VAX-LOC-86, VAX-LIB-86, VAX-OH-86, Package
VVSPLM86	VAX-PLM-86 Package
VVSPAS86	VAX-PASCAL-86 Package
VVSC86	VAX-C-86 Package
VVSFORT86	VAX-FORTRAN-86 Package
MVVSASM86	MICROVAX ASM86 Package
MVVSPLM86	MICROVAX PLM86 Package
MVVSC86	MICROVAX C86 Package
MVVSFORT86	MICROVAX FORTRAN 86 Package
MVVSPAS86	MICROVAX PASCAL-86 Package

REQUIRES SOFTWARE LICENSE

- **Library to Support Floating Point Arithmetic in Pascal-86, PL/M-86, FORTRAN-86, ASM-86, and iC-86**
- **Decimal Conversion Library Supports Binary-Decimal Conversions**
- **Supports Proposed IEEE Floating Point Draft 8.0 Standard for High Accuracy and Software Portability**
- **Common Elementary Function Library Provides Trigonometric, Logarithmic and Other Useful Functions**
- **Error-Handler Module Simplifies Floating Point Error Recovery**

The 8087 Support Library provides iC-86, Pascal-86, FORTRAN-86, PL/M-86 and ASM-86 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 8087 Support Library can be bound with PL/M-86 and ASM-86 user code to do this. The 8087 Support Library supports Draft 8.0 of the IEEE Floating Point Standard page 754. Consequently, by using this Library, the user saves software development time and the software investment is maintained.

The 8087 Support Library consists of the common elementary function library (CEL87.LIB), the decimal conversion library (DC87.LIB), the emulator interface library E8087.LIB, the error handler module (EH87.LIB) and interface libraries (8087.LIB, NUL87.LIB).

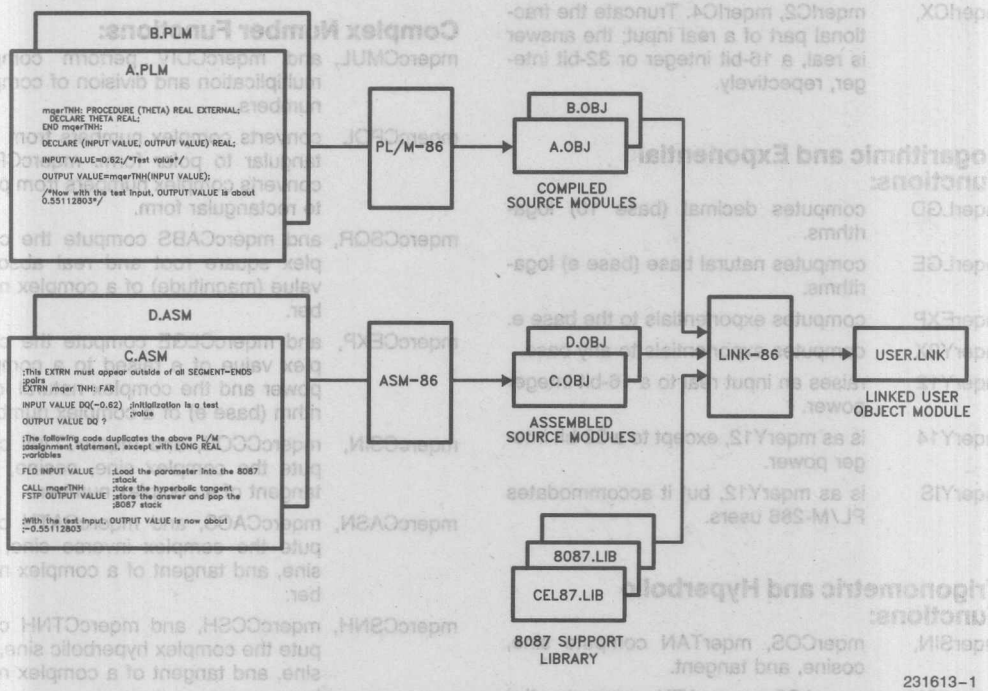


Figure 1. Use of 8087 Support Library with PL/M-86 and ASM-86

October 1988
Order Number: 231613-002

CEL87.LIB
THE COMMON ELEMENTARY FUNCTION LIBRARY

FUNCTIONS

CEL87.LIB contains commonly used floating point functions. It is used along with the 8087 numeric coprocessor. It provides a complete package of elementary functions, giving valid results for all appropriate inputs. Following is a summary of CEL87 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX, mqerIE2, and mqerIE4. Round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqerIAX, mqerIA2, mqerIA4. Round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqerICX, mqerIC2, mqerIC4. Truncate the fractional part of a real input; the answer is real, a 16-bit integer or 32-bit integer, repectively.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.

mqerLGE computes natural base (base e) logarithms.

mqerEXP computes exponentials to the base e.

mqerY2X computes exponentials to any base.

mqerY12 raises an input real to a 16-bit integer power.

mqerY14 is as mqerY12, except to a 32-bit integer power.

mqerYIS is as mqerY12, but it accommodates PL/M-286 users.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.

mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.

mqerAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

Other Functions (of real variables):

mqerDIM is FORTRAN's positive difference function.

mqerMAX returns the maximum of two real inputs.

mqerMIN returns the minimum of two real inputs.

mqerSGH combines the sign of one input with the magnitude of the other input.

mqerMOD computes a modulus, retaining the sign of the dividend.

mqerRMD computes a modulus, giving the value closest to zero.

Complex Number Functions:

mqercCMUL, and mqercCDIV perform complex multiplication and division of complex numbers.

mqercCPOL converts complex numbers from rectangular to polar form. mqercCREC converts complex numbers from polar to rectangular form.

mqercCSQR, and mqercCABS compute the complex square root and real absolute value (magnitude) of a complex number.

mqercCEXP, and mqercCLGE compute the complex value of e raised to a complex power and the complex natural logarithm (base e) of a complex number.

mqercCSIN, mqercCCOS, and mqercCTAN compute the complex sine, cosine, and tangent of a complex number.

mqercCASN, mqercCACS, and mqercCATN compute the complex inverse sine, cosine, and tangent of a complex number.

mqercCSNH, mqercCCSH, and mqercCTNH compute the complex hyperbolic sine, cosine, and tangent of a complex number.

mqercCACH, mqercCASH, and mqercCATH compute the comples inverse hyperbolic sine, cosine, and tangent of a complex number.

mqercCC2C, mqercCR2C, mqercCC2R, mqercCCI2, mqercCCI4, and mqercCCIS return complex values of complex (or real) values raised to complex (real, short integer, or long integer) values.

DC87.LIB
THE DECIMAL CONVERSION LIBRARY

DC87.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN__DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 8087. Because there are so many output formats for floating point numbers, mqcBIN__DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC__BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure mqcDECLOW__BIN is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG__TEMP, mqcSHORT__TEMP, mqcTEMP__LONG, and mqcTEMP__SHORT convert floating point numbers between the longest binary format, TEMP__REAL, and the shorter formats.

EH87.LIB
THE ERROR HANDLER LIBRARY

EH87.LIB is a library of five utility procedures for writing trap handlers. Trap handlers are called when an unmasked 8087 error occurs.

The 8087 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE draft standard options not directly supported by the chip. The three such extensions to the 8087 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 8087, and also identifies what function called the trap handler, and returns available arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception. By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE draft standard features are useful for diagnostic work.

ENCODE is called near the end of the trap handler. It restores the state of the 8087 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH87.LIB can be accomplished with a single call to FILTER.

8087.LIB, NUL87.LIB, E8087.LIB INTERFACE LIBRARIES

E8087.LIB, 8087.LIB and NUL87.LIB libraries configure a user's application program for the run-time environment; running with the 8087 component or without floating point arithmetic, respectively.

FULL 8087 EMULATOR

The Full 8087 Emulator is a 16-kilobyte object module that is linked to the application program for floating-point operations. Its functionality is identical to the 8087 chip, and is ideal for prototyping and debugging floating-point applications. The Emulator is an alternative to the use of the 8087 chip, although the latter executes floating-point applications up to 100 times faster than an 8086 with the 8087 Emulator. Furthermore, since the 8087 is a "coprocessor," use of the chip will allow many operations to be performed in parallel with the 8086.

SPECIFICATIONS

Operating Environment

Intel Microcomputer Development Systems (Series III, Series IV)

Documentation Package

8087 Support Library Reference Manual

ORDERING INFORMATION

8087 Support Library is included in ASM-86 Assembler package on the following hosts.

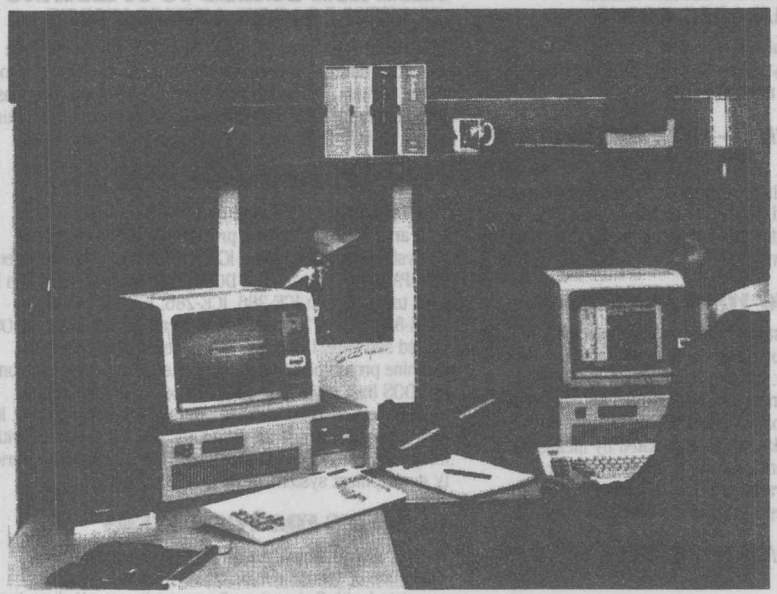
Part Number	Description
D86ASM86NL	ASM-86 Assembler for PC XT or AT System (or compatible) running DOS 3.0 or higher.
VVSASM86	ASM-86 Assembler for VAX/VMS.
MVVSASM86	ASM-86 Assembler for Micro VAX/VMS.

Requires Software License

SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

iPAT™ PERFORMANCE ANALYSIS TOOL



REAL-TIME SOFTWARE ANALYSIS FOR THE 8086/88, 80186/188, 80286, AND 80386

Intel's iPAT™ Performance Analysis Tool enables OEMs developing applications based on the 8086/88, 80186/188, 80286, or 80386 microprocessors to analyze real-time software execution in their prototype systems at speeds up to 20 MHz. Through such analysis, it is possible to speed-tune applications with real-time data, optimize use of operating systems (such as Intel's iRMX® II Real-Time Multitasking Executive for the 80286 and 80386, and iRMK™ Real-Time Multitasking Kernel for the 80386), characterize response characteristics, and determine code execution coverage by real-time test suites. Analysis is performed symbolically, non-intrusively, and in real-time with 100% sampling in the microprocessor prototype environment. iPAT supports analysis of OEM-developed software built using 8086, 80286, and 80386 assemblers and compilers supplied by Intel and other vendors.

All iPAT Performance Analysis Tool products are serially linked to DOS computer systems (such as IBM* PC AT, PC XT, and PS/2* Model 80) to host iPAT control and graphic display software. Several means of access to the user's prototype microprocessor system are supported. For the 80286 (real and protected mode), a 12.5 MHz iPAT-286 probe can be used with the iPATCORE system. For the 8086/88 (MAX MODE designs only), a 10 MHz iPAT-88 probe can be used with the iPATCORE system. iPATCORE systems also can be connected to sockets provided on the ICE™-286 and ICE-186 in-circuit emulators, or interfaced to I²ICE™ in-circuit emulators with probes supporting the 8086/88, 80186/188, or 80286. The 20 MHz iPAT™-386™ probe, also supported by the common iPATCORE system, can be operated either in "piggyback" fashion connected to an Intel ICE in-circuit emulator for the Intel386™, or directly connected to a prototype system independent of an ICE. iPAT-386 supports all models of 80386 applications anywhere in the lowest 16 Megabytes of the 80386 linear address space.

iPAT FEATURES

- Up to 20 MHz real-time analysis
- Histograms and analysis tables
- Performance profiles of up to 125 partitions
- Code execution coverage over up to 252K
- Hardware or software interrupt analysis
- Simple use with function keys and graphics
- Use with or without Intel ICEs

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© Intel Corporation 1988

June, 1988
Order Number: 280786-002

FEATURES

MOST COMPLETE REAL-TIME ANALYSIS AVAILABLE TODAY

iPAT Performance Analysis Tools use in-circuit probes containing proprietary chip technology to achieve full sampling in real-time non-intrusively.

MEETS THE REAL-TIME DESIGNER'S NEEDS

The iPAT products include support for interactions between real-time software and hardware interrupts, real-time operating systems, "idle" time, and full analysis of real-time process control systems.

SPEED-TUNING YOUR SOFTWARE

By examining iPAT histogram and tabular information about procedure usage (including or not including their interaction with other procedures, hardware, operating systems, or interrupt service routines) for critical functions, the software engineer can quickly pinpoint trouble spots. Armed with this information, bottlenecks can be eliminated by means such as changes to algorithms, recoding in assembler, or adjusting system interrupt priorities. Finally, iPAT can be used to prove the acceptability of the developer's results.

EFFICIENCY AND EFFECTIVENESS IN TESTING

With iPAT code execution coverage information, product evaluation with test suites can be performed more effectively and in less time. The evaluation team can quickly pinpoint areas of code that are executed or not executed under real-time conditions. By this means, the evaluation team can substantially remove the "black box" aspect of testing and assure 100% hits on the software under test. Coverage information can be used to document testing at the module, procedure, and line level. iPAT utilities also support generation of instruction-level code coverage information.

ANALYSIS WITH OR WITHOUT SYMBOLICS

If your application is developed with "debug" symbolics generated by Intel 8086, 80286, or 80386 assemblers and compilers, iPAT can use them—automatically. Symbolic names also can be defined within the iPAT environment, or conversion tools supplied with the iPAT products can be used to create symbolic information from virtually any vendor's map files for 8086, 80286, and 80386 software tools.

REAL OR PROTECTED MODE

iPAT supports 80286 and 80386 protected mode symbolic information generated by Intel 80286 and 80386 software tools. It can work with absolute addresses, as well as base:offset or selector:offset references to partitions in the prototype system's execution address space.

FROM ROM-LOADED TO OPERATING SYSTEM LOADED APPLICATIONS

The software analysis provided by iPAT watches absolute execution addresses in-circuit in real time, but also supports use of various iPAT utilities to determine the load locations for load-time located software, such as applications running under iRMXII, DOS, Microsoft Windows*, or MS*-OS/2.

USE STANDALONE OR WITH ICE

The iPAT-386, iPAT-286, and iPAT-86/88 probes, together with an iPATCORE system, provide standalone software analysis independent of an ICE (in-circuit emulator) system. The iPATCORE system and DOS-hosted software also can be used together with ICE-386, ICE-286, ICE-186, and I²ICE-86/88, 186/188, or 286 in-circuit emulators and DOS-hosted software. Under the latter scenario, the user can examine prototype software characteristics in real-time on one DOS host while another DOS host is used to supply input or test conditions to the prototype through an ICE. It also is possible to use an iPATCORE and I²ICE system with integrated host software on a single Intel Series III or Series IV development system, or on a DOS computer.

UTILITIES FOR YOUR NEEDS

Various utilities supplied with iPAT products support generation of symbolic information from map files associated with 3rd-party software tools, extended analysis of iPAT code execution coverage analysis data, and convenience in the working environment. For example, symbolics can be generated for maps produced by most software tools, instruction-level code execution information can be produced, and iRMXII-format disks can be read/written in DOS floppy drives to facilitate file transfer.

WORLDWIDE SERVICE AND SUPPORT

All iPAT Performance Analysis Tool products are supported by Intel's worldwide service and support. Total hardware and software support is available, including a hotline number when the need is there.

*Microsoft Windows and MS are trademarks of Microsoft Corporation. IBM and PS/2 are trademarks of International Business Machines. iPAT, ICE, iRMK, I2ICE, iRMX II, Intel386 are trademarks of Intel Corporation.

CONFIGURATION GUIDE

For all of the following application requirements, the iPAT system is supported with iPAT 2.0 (or greater) or iPAT/I²ICE 1.2 (or greater) host software, as footnoted.

Application Software	Option	iPAT Order Codes	Host System
80386 Embedded	#1	iPAT386DOS[1], iPATCORE	DOS
iRMK on 80386	#1	iPAT386DOS, iPATCORE	DOS
iRMXII OS-Loaded or Embedded on 386	#1	iPAT386DOS, iPATCORE	DOS
OS/2-Loaded on 386	#1	iPAT386DOS, iPATCORE	DOS
iRMXII OS-Loaded or Embedded	#1	iPAT286DOS, iPATCORE	DOS
80286 Embedded	#1	iPAT286DOS, iPATCORE	DOS
	#2	ICEPATKIT[2]	DOS
	#3	I²ICEPATKIT[3]	DOS
	#4	IIIPATD, iPATCORE[3]	DOS[4]
	#5	IIIPATB, iPATCORE[3]	Series III[4]
	#6	IIIPATC, iPATCORE[3]	Series IV[4]
DOS OS-Loaded 80286	#1	iPAT286DOS, iPATCORE	DOS
OS/2 OS-Loaded 80286	#1	iPAT286DOS, iPATCORE	DOS
80186/188 Embedded	#1	ICEPATKIT[2]	DOS
	#2	I²ICEPATKIT[3]	DOS
	#3	IIIPATD, iPATCORE[3]	DOS[4]
	#4	IIIPATB, iPATCORE[3]	Series III[4]
	#5	IIIPATC, iPATCORE[3]	Series IV[4]
DOS OS-Loaded 8086/88	#1	iPAT88DOS, iPATCORE	DOS
8086/88 Embedded	#1	iPAT88DOS, iPATCORE	DOS
	#2	I²ICEPATKIT[3]	DOS
	#3	IIIPATD, iPATCORE[3]	DOS[4]
	#4	IIIPATB, iPATCORE[3]	Series III[4]
	#5	IIIPATC, iPATCORE[3]	Series IV[4]

Notes:

1. Operable standalone or with ICE-386 (separate product; separate host). iPAT-386 probe connects directly to prototype system socket, or to optional 4″ probe-to-socket hinge cable (order code TA386A), or to ICE-386 probe socket.
2. Requires ICE-186 or ICE-286 in-circuit emulator system.
3. Requires I²ICE in-circuit emulator system.
4. Includes iPAT/I²ICE integrated software (iPAT/I²ICE 1.2 or greater), which only supports sequential iPAT and ICE operation on one host, rather than in parallel on two hosts (iPAT 2.0 or greater).

SPECIFICATIONS

HOST COMPUTER REQUIREMENTS

All iPAT Performance Analysis Tool products are hosted on IBM PC AT, PC XT, or PS/2 Model 80 personal computers, or 100% compatibles, and use a serial link for host-to-iPAT communications. At least a PC AT class system is recommended. The DOS host system must meet the following minimum requirements:

- 640K Bytes of Memory
- 360K Byte or 1.2M Byte floppy disk drive
- Fixed disk drive
- A serial port (COM1 or COM2) supporting 9600 baud data transfer
- DOS 3.0 or later
- IBM or 100% compatible BIOS

PHYSICAL DESCRIPTIONS

Unit	Width Inches	Width Cm.	Height Inches	Height Cm.	Length Inches	Length Cm.
iPATCORE	8.25	21.0	1.75	4.5	13.75	35.0
Power Supply	7.75	20.0	4.25	11.0	11.0	28.0
iPAT-386 probe	3.0	7.6	0.50	1.3	4.0	10.1
iPAT-286 probe	4.0	10.2	1.12	2.8	6.0	15.3
iPAT-86 probe	4.0	10.2	1.12	2.8	6.0	15.3
iPATCABLE (to ICE-186/286)	4.0	10.2	.25	.6	36.0	91.4
IIIPATB,C,D (I^2ICE board)	12.0	30.5	12.0	30.5	.5	1.3
Serial cables PC AT/XT PS/2					144.0	370.0

ELECTRICAL CONSIDERATIONS

The iPATCORE system power supply uses an AC power source at 100V, 120V, 220V, or 240V over 47Hz to 63Hz. 2 amps (AC) at 100V or 120V; 1 amp at 220V or 240V.

iPAT-386, iPAT-286 and iPAT-86/88 probes are externally powered, impose no power demands on the user's prototype, and can thus be used to analyze software activity through power down and power up of a prototype system. For ICE-386, ICE-286, ICE-186, and I^2ICE microprocessor probes, see the appropriate in-circuit emulator factsheets.

ENVIRONMENTAL SPECIFICATIONS

Operating Temperature: 10°C to 40°C (50°F to 104°F) ambient

Operating Humidity: Maximum of 85% relative humidity, non-condensing

I²ICE™ IN-CIRCUIT EMULATION SYSTEM

IN-CIRCUIT EMULATOR FOR THE 8086/80186/80286 FAMILY OF MICROPROCESSORS

The I²ICE™ In-Circuit Emulator is a high-performance, cost-effective debug environment for developing systems with the Intel 8086/80186/80286 family of microprocessors. With 10 MHz emulation, a window-oriented user interface, and compatibility with Intel's iPAT™ Performance Analysis Tool, the I²ICE Emulator gives you unmatched speed and control over all phases of hardware/software debug.

FEATURES

- Emulation speeds up to 10 MHz with 8086/88, 80186/188 and 80286 microprocessors
- 8087 and 80287 numeric coprocessor support
- Hosted on IBM PC AT*, AT BIOS, or compatibles
- ICEVIEW™ window-oriented user interface with pull-down menus and context-sensitive help
- Source and symbol display using all Intel languages
- 1K frame bus and execution trace buffer
- Symbolic debugging for flexible access to memory location and program variables
- Flexible breakpointing for quick problem isolation
- Memory expandable to 288K with zero wait states
- Worldwide service and support
- iPAT option for software speed tuning

intel

I2ICE, ICEVIEW, and iPAT are trademarks of Intel Corporation.

*IBM is a trademark of International Business Machines Corp.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1988

July, 1988
Order Number: 280800-001

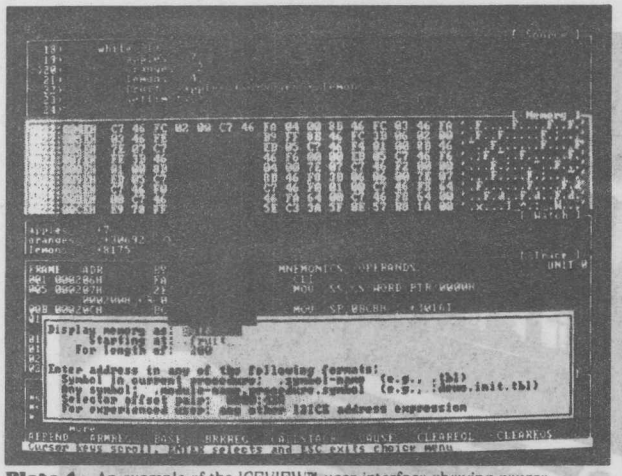


Plate 1. An example of the ICEVIEW™ user interface showing source, memory, watch, and trace.

ONE TOOL FOR THE ENTIRE DEVELOPMENT PROCESS

The I²ICE Emulator allows hardware and software design to proceed simultaneously, so you can develop software even before prototype hardware is available. With 32K of zero wait-state mappable memory (and an additional 256K with optional memory boards), you can use the I²ICE Emulator to debug at any stage of the development cycle: hardware development, software development, system integration or system test.

HIGH-SPEED, REAL-TIME EMULATION

The I²ICE Emulator delivers full-speed, real-time emulation at speeds up to 10 MHz. Based on Intel's exclusive microprocessor technology, the I²ICE Emulator matches each chip's electrical and timing characteristics without memory or interrupt intrusions, ensuring design accuracy and eliminating surprises. The performance of your prototype is the performance you can expect from your final product.

EASY-TO-USE ICEVIEW™ INTERFACE

The ICEVIEW interface makes the I²ICE Emulator easy to learn and use by providing easy access to application information and ICE functions. Pull-down menus and windows boost productivity for both new and experienced users. Multiple on-screen windows allow you to access the source display, execution trace, register, and other important information, all at the same time. You can watch the information change as you modify and step through your program. You can even customize window size and screen positions.

A command line interface is also available with syntax checking and context-sensitive prompts. ICEVIEW works with monochrome, CGA and the latest EGA color displays.

SYMBOLIC DEBUG SPEEDS DEVELOPMENT

The extensive debug symbolics generated by the Intel 8086 and 80286 assemblers and compilers can increase your development productivity. Symbolics with automatic formatting are available for all primitive types, regardless of whether the variables are globals, locals (stack-resident) or pointers. The virtual symbol table supports all symbolics, even in very large programs. Aliasing can be used to reduce keystrokes and save time.

POWERFUL BREAK AND TRACE CAPABILITY FOR FAST PROBLEM ISOLATION

The I²ICE Emulator allows up to eight simultaneous break/trace conditions to be set (four execution, four bus), a timesaver when solving hardware/software integration problems. Break and trace points can be set on specified line numbers, on procedures, or on symbolic data events, such as writing a variable to a value or range of values. You can break or trace on specific hardware events, such as a read or write to a specific address, data or I/O port, or on a combination of events.

MULTIPROCESSOR, PROTECTED MODE, AND COPROCESSOR SUPPORT

Up to four I²ICE systems can be linked and controlled simultaneously from one PC host, enabling you to debug multiprocessor systems. The I²ICE Emulator with an 80286 probe supports all 80286 protected mode capabilities. It also supports the 8087 and 80287 numeric coprocessors.

iPAT™ FOR SOFTWARE PERFORMANCE AND CODE COVERAGE ANALYSIS

The I²ICE Emulator interfaces to Intel's iPAT Performance Analysis Tool for examining software execution speeds and code coverage in real time. iPAT displays critical performance data about your code in easy-to-understand histograms and tables. Elusive bottlenecks are readily seen, allowing you to focus your attention to get the most performance out of your product.

iPAT also performs code execution coverage, letting you perform product evaluations faster and more effectively. iPAT pinpoints areas in your code either executed or not executed according to specific conditions, taking the guesswork out of software evaluations.

EASY INTERFACE TO EXTERNAL INSTRUMENTS

The I²ICE system includes external emulation clips and software support for setting breakpoints, tracepoints and arm/disarm conditions on external events, making it easy to connect external logic analyzers and signal generators. You can debug complex hardware/software interactions with a high level of productivity.

WORLDWIDE SERVICE AND SUPPORT

The I²ICE Emulator is supported by Intel's worldwide service and support organization. In addition to an extended warranty, you can choose from hotline support, on-site systems engineering assistance, and a variety of hands-on training workshops.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC/AT or 100% PC AT BIOS compatible
DOS 3.1 or later
640K bytes of memory
360K bytes or 1.2 MB floppy disk drive
Hard disk drive
Monochrome, CGA or EGA monitor (EGA recommended)

PHYSICAL DESCRIPTION

Unit	Width		Height		Length	
	cm	in	cm	in	cm	in
I²ICE chassis	43.2	17.0	21.0	8.25	61.3	24.13
Probe base	21.6	8.5	7.6	3.0	25.4	10.0

Host/chassis cable 15 ft. (4.6 m)

ELECTRICAL CHARACTERISTICS

90-132 V or 180-264 V (selectable)
47-63 Hz
12 amps (AC)

ENVIRONMENTAL SPECIFICATIONS

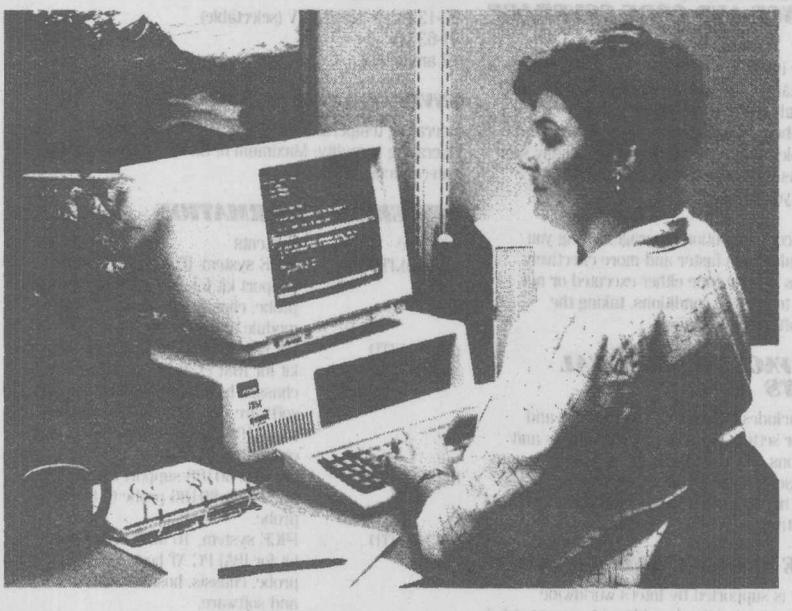
Operating temperature: 0-40°C (32-104°F) ambient
Operating humidity: Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

Kit Code	Contents
pIII010KITD	I²ICE system 10 MHz 8086/8088 support kit for IBM PC host. Includes probe, chassis, and host interface module and software.
pIII11KITD	I²ICE system 10 MHz 80186 support kit for IBM PC host. Includes probe, chassis, host interface module and software. Note: For 80188 support, the III98 option below must also be ordered.
III98	10 MHz 80188 support conversion kit to convert 80186 probe to 80188 probe.
pIII212KITD	I²ICE system, 10 MHz 80286 support kit for IBM PC AT host. Includes probe, chassis, host interface module and software.
III010PATC86D	I²ICE system 10 MHz 8086/8088 support kit with iPAT Performance Analysis Tool for PC AT host. Includes I²ICE probe, chassis, host interface module, iPAT tool option, cables and software. Also includes iC-86 compiler, 86 Macro Assembler, utilities, and AEDIT text editor.
III11PATC86D	As above for 10 MHz 80186 support.
III212PATC86D	As above for 10 MHz 80286 support. Note: C-286 and RLL-286 and ASM-286 must be ordered separately.
954D	I²ICE PC AT host software. Includes ICEVIEW™ windowed human interface.

Note: I²ICE probes, chassis, software, cables and iPAT options are available separately.

AEDIT SOURCE CODE AND TEXT EDITOR



PROGRAMMER SUPPORT

AEDIT is a full-screen text editing system designed specifically for software engineers and technical writers. With the facilities for automatic program block indentation, HEX display and input, and full macro support, AEDIT is an essential tool for any programming environment. And with AEDIT, the output file is the pure ASCII text (or HEX code) you input—no special characters or proprietary formats.

Dual file editing means you can create source code and its supporting documents at the same time. Keep your program listing with its errors in the background for easy reference while correcting the source in the foreground. Using the split-screen windowing capability, it is easy to compare two files, or copy text from one to the other. The DOS system-escape command eliminates the need to leave the editor to compile a program, get a directory listing, or execute any other program executable at the DOS system level.

There are no limits placed on the size of the file or the length of the lines processed with AEDIT. It even has a batch mode for those times when you need to make automatic string substitutions or insertions in a number of separate text files.

AEDIT FEATURES

- Complete range of editing support—from document processing to HEX code entry and modification
- Supports system escape for quick execution of PC-DOS System level commands
- Full macro support for complex or repetitive editing tasks
- Hosted on PC-DOS and RMX operating systems
- Dual file support with optional split-screen windowing
- No limit to file size or line length
- Quick response with an easy to use menu driven interface
- Configurable and extensible for complete control of the editing process

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

September, 1988
Order Number: 280804-002

© Intel Corporation 1988

FEATURES

POWERFUL TEXT EDITOR

As a text editor, AEDIT is versatile and complete. In addition to simple character insertion and cursor positioning commands, AEDIT supports a number of text block processing commands. Using these commands you can easily move, copy, or delete both small and large blocks of text. AEDIT also provides facilities for forward or reverse string searches, string replacement and query replace.

AEDIT removes the restriction of only inserting characters when adding or modifying text. When adding text with AEDIT you may choose to either insert characters at the current cursor location, or over-write the existing text as you type. This flexibility simplifies the creation and editing of tables and charts.

USER INTERFACE

The menu-driven interface AEDIT provides makes it unnecessary to memorize long lists of commands and their syntax. Instead, a complete list of the commands or options available at any point is always displayed at the bottom of the screen. This makes AEDIT both easy to learn and easy to use.

FULL FLEXIBILITY

In addition to the standard PC terminal support provided with AEDIT, you are able to configure AEDIT to work with almost any terminal. This along with user-definable macros and full adjustable tabs, margins, and case sensitivity combine to make AEDIT one of the most flexible editors available today.

MACRO SUPPORT

AEDIT will create macros by simply keeping track of the command and text that you type, "learning" the function the macro is to perform. The editor remembers your actions for later execution, or you may store them in a file to use in a later editing session.

Alternatively, you can design a macro using AEDIT's powerful macro language. Included with the editor is an extensive library of useful macros which you may use or modify to meet your individual editing needs.

TEXT PROCESSING

For your documentation needs, paragraph filling or justification simplifies the chore of document formatting. Automatic carriage return insertion means you can focus on the content of what you are typing instead of how close you are to the edge of the screen.

SERVICE, SUPPORT, AND TRAINING

Intel augments its development tools with a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hot-line support; and software and hardware maintenance contracts. This full line of services will ensure your design success.

SPECIFICATIONS

HOST SYSTEM

AEDIT for PC-DOS has been designed to run on the IBM* PC XT, IBM PC AT, and compatibles. It has been tested and evaluated for the PC-DOS 3.0 or greater operating system.

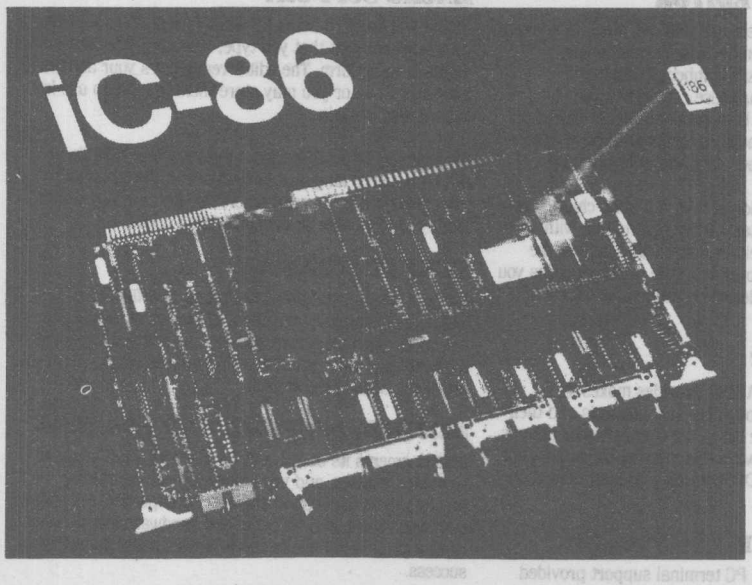
Versions of AEDIT are available for the iRMX™-86 and RMX II Operating System.

ORDERING INFORMATION

D86EDINL	AEDIT Source Code Editor Release 2.2 for PC-DOS with supporting documentation
122716	AEDIT-DOS Users Guide
122721	AEDIT-DOS Pocket Reference
RMX864WSU	AEDIT for iRMX-86 Operating System
R286EDI286EU	AEDIT for iRMX II Operating System

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

INTEL iC-86 C COMPILER

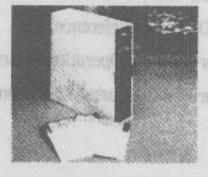


INTEL iC-86 R4.0 COMPILER

Intel's iC-86 R4.0 is a new generation C compiler for the 8086/186 family of microprocessors, providing unparalleled performance for embedded microprocessor designs. In addition to outstanding execution speed, Intel's iC-86 also offers low memory consumption, ROMability, and easy debug.

iC-86 R4.0 COMPILER FEATURES

- State-of-the-art code generation technology
- Built-in functions for automatic machine code generation
- ROMable code and libraries
- Outstanding optimization
- Integrated debugging with Intel ICE™ and I²ICE™
- Compliance with draft ANSI standard
- Supports multiple memory models: Small, Medium, Compact, and Large
- Linkable with other Intel 8086 languages such as ASM-86 and PL/M-86
- ROMable and reentrant libraries
- Ability to mix memory models with "near" and "far" pointers
- Compatible with other C compilers and PL/M providing both standard C and PL/M calling conventions



intel

ICE and I²ICE are trademarks of Intel Corporation.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© Intel Corporation 1988

February, 1988
Order Number: 280787-003

BUILT-IN FUNCTIONS

iC-86 R4.0 is loaded with built-in functions that directly generate machine code within the C language. iC-86's built-ins eliminate the need for in-line assembly language programming by allowing you to program in a high-level language all the time, speeding software development and simplifying software maintenance. The built-ins also allow you to develop highly optimized code by extending the compiler instruction set—with built-ins you can enable or disable interrupts and directly control hardware I/O without having to exit C for assembler. This means you can write high performance software for real time applications without having to keep track of every architectural detail, as you would in assembly language. For example, to generate an INT instruction, you simply type:

```
causeinterrupt(number)
```

Or, the following iC-86 instruction will cause the processor to come to a halt with interrupts enabled:

```
halt( )
```

EMBEDDED COMPONENT SUPPORT

iC-86 was designed specifically for embedded microprocessor applications. iC-86 produces ROMable code which can be loaded directly into target systems via Intel ICE emulators and debugged *without modification* for fast, easy, development and debugging.

HIGHLY OPTIMIZED

iC-86 is based on Intel's latest code generation techniques for developing high-performance applications, and has been optimized for developing embedded applications. Four levels of optimization are available. Important optimization features include a jump optimizer and improved register manipulation via register history. In addition, the PL/M calling convention will improve performance significantly. An example of the optimization in iC-86 R4.0 is its outstanding performance on the Dhrystone benchmark. Using a Compaq 386, iC-86 produced the following results:

	Microsoft 4.0	Microsoft 5.0	Intel iC-86
Execution Speed (dhry/sec)	3333	3369	3571

RUN-TIME SUPPORT

STDIO run-time libraries for iC-86 are targeted to a generic POSIX interface, with documentation provided for interfacing with your embedded target system. This means you can easily retarget the libraries for use in your target application, regardless of operating system. These libraries support the complete draft ANSI standard. For iC-86 versions on DOS, Intel provides the interface between the STDIO libraries and the DOS operating system. This allows you to develop, test, and debug your embedded application code on DOS, or write applications directly for DOS.

INTEGRATED DEBUG TOOLS

iC-86 has been designed to work with Intel's ICE family of in-circuit emulators (I²ICE, ICE-186, ICE-286, and ICE-386) and performance analysis tools (iPAT). Intel software debuggers, linkers, locators, and other software development tools. In addition to the object records required for program execution, iC-86 object code contains detailed debug records that describe the actual symbols and variable names you defined in your source code. A complete listing file can also be produced. Intel's "integration by design" of all development tools, including iC-86, will speed the development of your embedded microprocessor applications. Figure 1 illustrates the steps in going from C source code to PROM- or ICE-loadable object code with iC-86.

SERVICE AND SUPPORT

Intel's development tools are backed by our worldwide service and support organization, which is set up to deal with problems encountered by embedded component designers. Our field application experts get you up and running quickly, and our hands-on training workshops ensure that you have a thorough understanding of how our tools work. Intel compilers come with 90 days of technical support, troubleshooting guides, application newsletters, and optional support contracts.

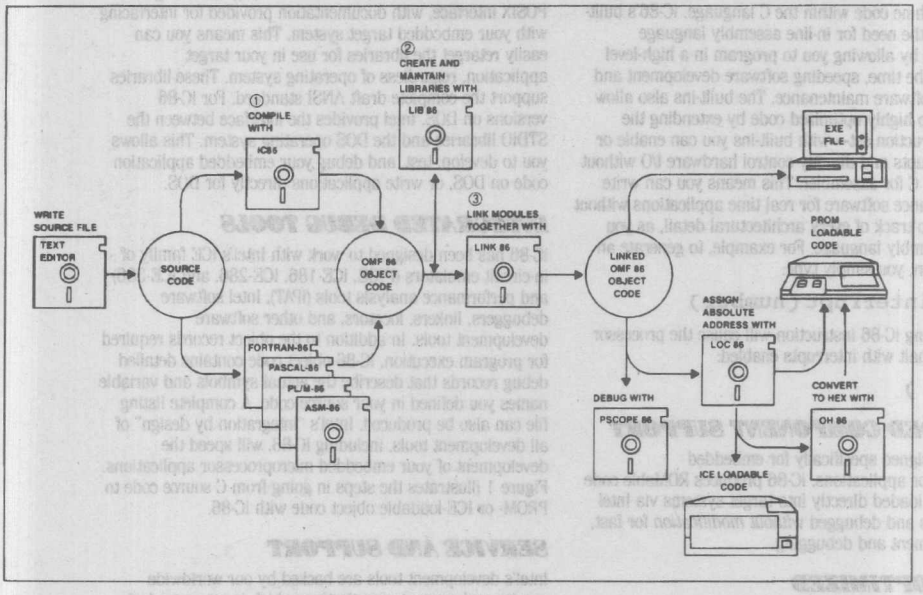


Figure 1. The Application Development Process

SPECIFICATIONS

ENVIRONMENT

Hardware requirements:	IBM PC XT or AT (or 100% compatible) running DOS 3.0 or greater
Memory requirements:	512K Bytes of RAM, hard disk strongly recommended
Media:	5¼" DS/DD DOS diskettes

STANDARDS

iC-86 complies with the 1986 X3J11 ANSI draft proposal for the C programming language

ORDERING INFORMATION

Order Code	Description
D86C86NL	DOS hosted iC-86

ICE™-186 IN-CIRCUIT EMULATOR

HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-186 emulator delivers real-time emulation for the 80C186 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C186 microprocessor. The ICE-186 emulator provides real time, full speed emulation in a user's system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-186 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's iC86, PL/M86, Pascal 86 and Fortran 86 to accelerate development schedules.

The ICE-186 emulator supports a subset of the 80C186 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-186 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-186 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C186 microprocessor.

ICE™-186 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Frames Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1988

September, 1988
Order Number: 280726-003

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-186 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C186 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one instruction at a time or in preset instruction blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-186 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-186 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-186 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-186 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for the Intel 80186/80188 component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-186 commands resulting in a consistent debug environment.

SUPPORTS FAST BREAKS

"Fastbreaks" is a feature which allows the emulation processor to halt, access memory, and return to emulation as quickly as possible. A fastbreak never takes more than 5625 clock cycles (most types of fastbreaks are considerably less). This feature is particularly useful in embedded applications.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-186 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-186 emulator has an external 60 pin connector for iPAT.

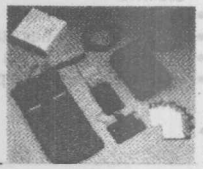
BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-186 to provide detailed timing of specific events. The ICE-186 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-186 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.

Note: This emulator does not support use of the 8087.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-186 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-186 Emulator consists of the following components:

	Width		Height		Length	
Unit	**Inches**	**Cm.**	**Inches**	**Cm.**	**Inches**	**Cm.**
Emulator Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	7.60	19.00	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable, Pice					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power Accessory	4.30	10.90	.60	1.50	6.70	17.00
CPA Power Cable					9.00	22.90

ELECTRICAL CONSIDERATIONS

Icc 1050mA
I_{IH} 70μA Max.
I_{IL} – 1.5mA Max
I_{OH} – 1.0mA Max.

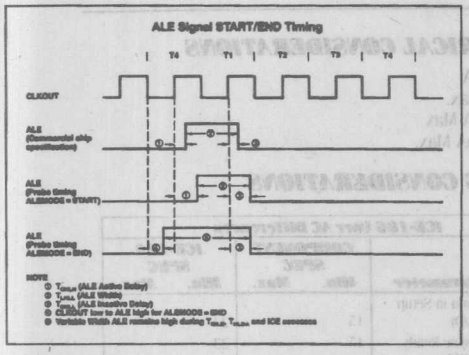
TIMING CONSIDERATIONS

ICE-186 User AC Differences

Symbol	Parameter	COMPONENT SPEC Min.	COMPONENT SPEC Max.	ICE-186 SPEC Min.	ICE-186 SPEC Max.
T_{DVCL}	Data in Setup (/D)	15		24	
T_{ARYCH}	Async Ready (ARDY) Resolution Transition Setup Time	15		23	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time	15		25	
T_{HVCL}	HOLD Setup	15		32	
T_{INVCH}	NMI	15		32	
	/TEST	15		31	
	INTR,TIMERIN Setup Time	15		17	
T_{INVCL}	DRQ0, DRQ1, Setup Time	15		19	
T_{CLAZ}	Address Float Delay				
	READ Cycles	T_{CLAX}	25	5	36
	INTA cycles	T_{CLAX}	25	0	25
	HLDA	T_{CLAX}	25	10	50
T_{LHLL}	ALE Width (min)	T_{CLCL}-30		T_{CLCL}-32	
T_{CLLH}	ALE Active Delay*		25		42
(N/A)	CLKOUT Low to ALE Active**		(N/A)		19
T_{CLLL}	ALE Inactive Delay		25		40
T_{LLAX}	Address Hold to ALE Inactive (min)	T_{CHCL}-15		T_{CHCL}-28	
T_{CVCTX}	Control Inactive Delay	5	37	1	40
T_{AZRL}	Address Float to /RD Active	0		-30	
T_{AVLL}	Address Valid to ALE Low (min)	T_{CLCH}-15		T_{CLCH}-19	
T_{CHSV}	Que Status Delay		28		35
T_{DXDL}	/DEN Inactive to DT /R Low	0		-7	
T_{CLCO}	CLKIN to CLKOUT Skew		21		37

Consult User Guide for Additional Specifications.

* Applies only when the ALEMODE variable is set to START.
** Applies only when the ALEMODE variable is set to END.



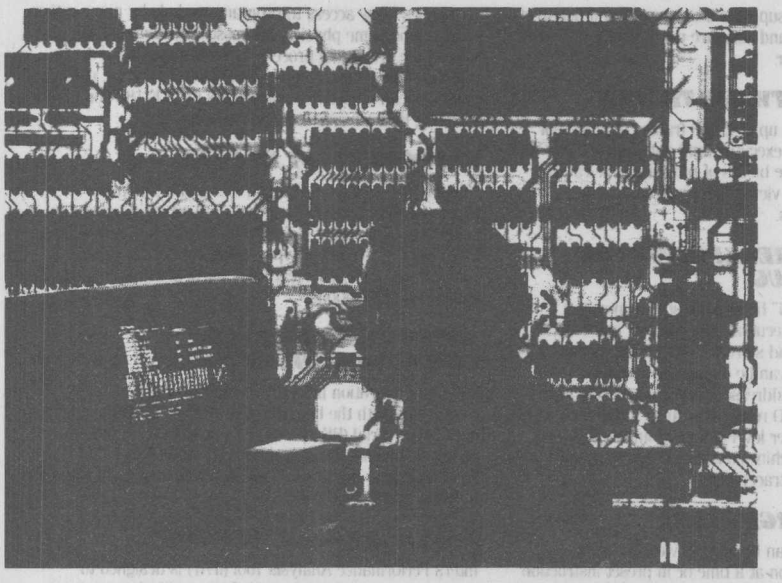
ENVIRONMENTAL SPECIFICATIONS

Operating Temperature 10°C to 40°C Ambient
Storage Temperature –40°C to 70°C

ORDERING INFORMATION

ICE186	ICE-186 System including ICE software (Requires DOS 3.XX PC AT with Above Board)
ICE 186AB	ICE 186 with Above Board included
ICE186IPAT	ICE-186 System including ICE S/W packages and the iPAT system (Requires DOS 3.XX PC AT with Above Board)
D86ASM86NL	86 macro assembler 86 builder/binder/ mapper utilities for DOS 3.XX.
D86C86NL	86 C compiler and run time libraries for DOS 3.XX.
D86PAS86NL	86 Pascal Compiler for DOS 3.XX.
D86PLM86NL	86 PL/M compiler for DOS 3.XX.
D86FOR86NL	86 Fortran compiler for DOS 3.XX.
ICEPAT KIT	iPAT Kit (Performance Analysis Tool) for ICE 186
ICEXONCE	Adapter for on-circuit emulation
ICEXLCC	Adapter for LCC component
ICEXPGA	Adapter for PGA component

ICE™-188 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-188 emulator delivers real-time emulation for the 80C188 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C188 microprocessor. The ICE-188 emulator provides real time, full speed emulation in a user's system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-188 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's iC86, PL/M86, Pascal 86 and Fortran 86 to accelerate development schedules.

The ICE-188 emulator supports a subset of the 80C188 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-188 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-188 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C188 microprocessor.

ICE™-188 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Frames Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

September, 1988
Order Number: 280986-003

© Intel Corporation 1988

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-188 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C188 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "IF-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one instruction at a time or in preset instruction blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-188 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-188 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-188 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-188 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for the Intel 80186/80188 component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-188 commands resulting in a consistent debug environment.

SUPPORTS FAST BREAKS

"Fastbreaks" is a feature which allows the emulation processor to halt, access memory, and return to emulation as quickly as possible. A fastbreak never takes more than 5625 clock cycles (most types of fastbreaks are considerably less). This feature is particularly useful in embedded applications.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-188 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-188 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-188 to provide detailed timing of specific events. The ICE-188 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-188 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.

Note: This emulator does not support use of the 8087.

SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-188 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-188 Emulator consists of the following components:

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
Emulator Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	7.60	19.00	1.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Plcc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power Accessory	4.30	10.90	.60	1.50	6.70	17.00
CPA Power Cable					9.00	22.90

ELECTRICAL CONSIDERATIONS

Icc 1050mA
I_{IH} 70μA Max.
I_{IL} – 1.5mA Max
I_{OH} – 1.0mA Max.

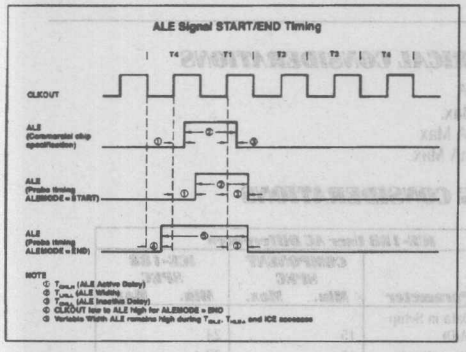
TIMING CONSIDERATIONS

ICE-188 User AC Differences

Symbol	Parameter	COMPONENT SPEC Min.	COMPONENT SPEC Max.	ICE-188 SPEC Min.	ICE-188 SPEC Max.
T_{DVCL}	Data in Setup (MD)	15		24	
T_{ARYCH}	Async Ready (ARDY) Resolution Transition Setup Time	15		23	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time	15		25	
T_{HVCL}	HOLD Setup	15		32	
T_{INVCH}	NMI	15		32	
	/TEST	15		31	
	INTR,TIMERIN Setup Time	15		17	
T_{INVCL}	DRQ0, DRQ1 Setup Time	15		19	
T_{CLAZ}	Address Float Delay				
	READ Cycles	T_{CLAX}	25	5	36
	INTA cycles	T_{CLAX}	25	0	25
	HLDA	T_{CLAX}	25	10	50
T_{LHLL}	ALE Width (min)	T_{CLCL}-30		T_{CLCL}-32	
T_{CHLH}	ALE Active Delay*		25		42
(N/A)	CLKOUT Low to ALE Active**		(N/A)		19
T_{CHLL}	ALE Inactive Delay		25		40
T_{CLAX}	Address Hold to ALE Inactive (min)	T_{CHCL}-15		T_{CHCL}-28	
T_{CVCTX}	Control Inactive Delay	5	37	1	40
T_{AZRL}	Address Float to /RD Active	0		– 30	
T_{AVLL}	Address Valid to ALE Low (min)	T_{CLLH}-15		T_{CLLH}-19	
T_{CHSV}	Que Status Delay		28		35
T_{DXDL}	/DEN Inactive to DT /R Low	0		– 7	
T_{CICO}	CLKIN to CLKOUT Skew		21		37

Consult User Guide for Additional Specifications.

*Applies only when the ALEMODE variable is set to START.
**Applies only when the ALEMODE variable is set to END.



ORDERING INFORMATION

ICE188	ICE-188 System including ICE software (Requires DOS 3.XX PC AT with Above Board)
ICE 188AB	ICE 188 with Above Board included
D86ASM86NL	86 macro assembler 86 builder/binder/ mapper utilities for DOS 3.XX.
D86C86NL	86 C compiler and run time libraries for DOS 3.XX.
D86PAS86NL	86 Pascal Compiler for DOS 3.XX.
D86PLM86NL	86 PL/M compiler for DOS 3.XX.
D86FOR86NL	86 Fortran compiler for DOS 3.XX.
ICEPAT KIT	iPAT Kit (Performance Analysis Tool) for ICE 188
ICEXONCE	Adapter for on-circuit emulation
ICENLCC	Adapter for LCC component
ICENPGA	Adapter for PGA component
UP188	User probe to convert ICE-186 to support 80C188 component

ENVIRONMENTAL SPECIFICATIONS

Operating Temperature 10°C to 40°C Ambient
Storage Temperature −40°C to 70°C

Index

Index

MCS®-96 INDEX

A

A-Bus, 1-02
A/D Converter, 1-33, 4-01, 4-09, 4-19
  Actual and Ideal Characteristic, 4-13
  Block Diagram, 4-09
  Commands, 1-34, 1-57
  Glossary, 4-15
  Interface Suggestions, 4-10
  Reference Voltages, 4-11
  Results, 1-34, 1-57
  Sample and Hold, 1-34
  Sampling Circuitry, 4-10
  Suggested Input Circuit, 4-11
  Transfer Function, 4-11
aa, 2-01
Absolute Error—A/D Converter, 4-16
Actual Characteristic—A/D Converter, 4-15, 4-16
AD0 (Address/Data 0), 4-20
AD0–AD15 (Address/Data 0–15), 1-08
ADD (Add words), 2-02
ADDB (Add bytes), 2-03
ADDC (Add words with Carry), 2-04
ADDCB (Add bytes with Carry), 2-04
Address 32-Bit Operands, 1-22
Address Decoding, 4-24
Address Valid Strobe Mode, 1-12
Address Valid with Write Strobe, 1-13
Address/Command/Data Bus, 4-31
Addressing Modes, 2-01
  Immediate, 1-16, 2-01
  Indirect, 2-01
  Indirect with Auto-Increment, 1-16
  Long-Indexed, 1-16, 2-01
  Register Direct, 2-01
  Short Indexed, 1-16, 2-01
  Stack Pointer Register, 1-17
  Zero Register, 1-17
$\overline{\text{ADV}}$ (Address Valid), 1-12
ALE (Address Latch Enable), 1-08, 1-11, 1-12, 4-09, 4-20, 4-26
ALU (Arithmetic Logic Unit), 1-02
Analog Inputs, 4-09
Analog Interface, 1-33
Analog Output, 4-17
Analog Reference Voltages—A/D Converter, 4-11
Analog-to-Digital Conversion (A/D), 1-33, 4-09
Analog/Digital Converter (D/A), 1-35
AND (Logical And Words), 2-05
ANDB (Logical And Bytes), 2-06
ANGND (Analog Ground), 1-33, 4-01, 4-11
Assembly Language Addressing Modes, 1-17
Auto Programming, 4-32

B

baop, 2-01
Baud Rates, 1-38, 1-57, 4-19
$\overline{\text{BEA}}$, 2-01
$\overline{\text{BHE}}$ (Bus High Enable), 1-08, 1-11, 4-09, 4-20, 4-24
Bit Operands, 1-14
bitno, 2-01
BR (Indirect), 2-07
Branch, 2-07
Break-Before-Make—A/D Converter, 4-16
breg, 2-01
Bus (System), 1-08, 1-40, 4-09, 4-20
Bus Control, 1-11
Bus Timings, 4-21
BUSWIDTH, 1-10, 4-24
Byte Operands, 1-14, 1-19

C

C Flag (see Carry Flag), 1-18
cadd, 2-01
CALL, 2-26, 2-40
CAM (Content Addressable Memory), 1-28
Carry Flag, 1-18, 2-41
CCB (Chip Configuration Byte), 1-10, 4-36
CCR (Chip Configuration Register), 1-10, 1-59, 4-21, 4-36
CEA, 2-01
Channel-To-Channel Matching—A/D Converter, 4-16
Characteristic—A/D Converter, 4-16
Chip Configuration Byte (see CCB), 1-10
Chip Configuration Register (see CCR), 1-10
Circuit
  Auto Configuration Byte Programming, 4-36
  Auto Programming, 4-32
  D/A, 4-18
  Gang Programming, 4-33
  Oscillator, 4-02
  Reset, 4-04
  Suggested A/D Input, 4-11
Clear Byte Instruction, 2-08
Clear Carry Flag, 2-08
Clear Word Instruction, 2-07
Clearing the HSO, 1-32
CLKOUT (Clock Out), 1-03, 1-09, 4-09, 4-21
CLR (Clear Word), 2-07
CLRB (Clear Byte), 2-08
CLRC (Clear Carry Flag), 2-08
CLRVT (Clear Overflow Trap), 2-09
CMP (Compare Words), 1-19, 2-09

C (Continued)
CMPB (Compare Bytes), 1-19, 2-10
Code—A/D Converter, 4-16
Code Center—A/D Converter, 4-16
Code Transition—A/D Converter, 4-16
Code Width—A/D Converter, 4-16
Compare Bytes Instruction, 2-10
Compare Words Instruction, 2-09
Complement Instruction, 2-35, 2-36
Condition Flags, 1-18
Conditional Jumps, 1-19
Configuration Byte Programming, 4-36
Content Addressable Memory (CAM), 1-28
CPU (Central Processing Unit), 1-01
Critical Regions, 1-26
Crosstalk—A/D Converter, 4-16

D
D-Bus, 1-02
D.C. Input Leakage—A/D Converter, 4-16
D/A Circuits, 4-18
D/A Converter, 4-17
D/A Digital/Analog Converter, 1-35
Data Program Command, 4-34
Data Program/Verify Signals, 4-35
Data Verify Command, 4-35
DEC (Decrement Word), 2-10
DECB (Decrement Byte), 2-11
Decrement Byte Instruction, 2-11
Decrement Word Instruction, 2-10
DI (Disable Interrupts), 1-25, 1-27, 2-11
Differential Non-Linearity—A/D Converter, 4-15, 4-16
Disabling The Watchdog, 1-43
DIV (Divide Integers), 2-12
DIVB (Divide Short-Integers), 2-12
Divide, 2-12, 2-13
DIVU (Divide Words), 2-13
DIVUB (Divide Bytes), 2-13
DJNZ (Dec and Jump if Not Zero), 2-14
Double-Word Operands, 1-15, 1-19
Drive and Interface Levels, 4-06

E
$\overline{EA}$ (External Access), 1-07, 1-10, 1-43, 4-01, 4-30, 4-38
EI (Enable Interrupts), 1-25, 1-27, 2-14
EPROM, 1-07
  Erasing, 4-39
  Lock, 1-13, 4-38
  Programming, 4-30
  Timings, 4-28
Erasing the 879XBH EPROM, 4-39
Examples
  Memory Systems, 4-25
  Port Reconstruction, 4-29
  Run-Time Programming, 4-37
  System Verification, 4-26

E
EXT (Sign Extend Integer into Long-Integer), 1-19, 2-15
EXTB (Sign Extend Short-Integer into Integer), 1-19, 2-15
External Clock Drive, 1-03, 4-2

F
Feedthrough—A/D Converter, 4-15, 4-16
FIFO (see HSI), 1-30
Flag Settings, 2-01
Flags, 1-18
FPAL-96, 1-19
Full-Scale Error—A/D Converter, 4-15, 4-16

G
Gang Programming
  Auto, 4-32
  Slave, 4-35
Generic Jumps and Calls, 2-01
  BH, 2-01
  BR, 2-01
  CALL, 2-01
  JH, 2-01
  LCALL, 2-01
  LJMP, 2-01
  SCALL, 2-01
  SJMP, 2-01
Global Interrupt Enable Bit (I Bit), 1-25

H
Hardware Connections minimum, 4-01, 4-04
High Speed Inputs (see HSI), 1-28, 1-29
High Speed Outputs (see HSO), 1-28, 1-31
HSI, 1-28
  Input Timings, 4-18
  Interrupts, 1-30
  Modes, 1-30, 1-57
  Status, 1-30, 1-57
HSO, 1-28, 1-31, 1-36, 4-17
  CAM, 1-31
  Clearing, 1-32
  Command Tag, 1-32
  Interrupts, 1-31
  Output Timings, 4-18
  Pins, 4-08
  Status, 1-31, 1-32

I
I Bit (Global Interrupt Enable Bit), 1-25
I/O Control Register 0 (IOC0), 1-41
I/O Control Register 1 (IOC1), 1-42
I/O Control Registers, 1-41
I/O Ports, 1-39
I/O Status Register 0 (IOS1), 1-42
I/O Status Register 1 (IOS2), 1-42

I (Continued)
I/O Timings, 4-18
Ideal A/D Characteristic—A/D Converter, 4-11, 4-16
Immediate Addressing, 1-16, 2-01
INC (Increment Word), 2-16
INCB (Increment Byte), 2-16
Increment Instruction, 2-16
Indirect Addressing, 1-15, 2-01
Indirect Shifts, 2-01
Indirect with Auto-Increment Addressing, 1-16
Input Ports, 1-39, 1-40, 4-08
Input Resistance—A/D Converter, 4-16
INST (Instruction), 4-20, 4-24
INST Line Usage, 4-24
Instruction Set, 1-18, 2-01
Instruction Summary, 1-20, 1-21, 1-52, 1-53
Integer Operands, 1-14, 1-19
Interface Levels, 4-06
Internal Memory
  EPROM, 1-07
  RAM, 1-04
  ROM, 1-07
Internal Ready Control, 1-13
Internal Timings, 1-03, 4-03
Interrupt, 1-23, 1-24
  Control, 1-25
  Disable, 2-11
  Enable, 2-14
  Flags, 1-18
  Global Disable, 1-25
  HSI, 1-30
  HSO, 1-32
  Mask Register, 1-25
  Nonmaskable, 1-04
  Pending Register, 1-25, 1-27
  Priorities, 1-25
  Serial Port (TI/RI), 1-37, 4-19
  Software Timers, 1-33
  Timer, 1-28
  Timing, 1-27
  Vectors, 1-25, 1-58
IOC0, 1-28, 1-41, 1-58
IOC1, 1-28, 1-42, 1-58
IOS0, 1-42, 1-58
IOS1, 1-28, 1-42, 1-58
IRC0, IRC1 Internal Ready Control, 1-31

J
JBC (Jump if Bit Clear), 2-17
JBS (Jump if Bit Set), 2-18
JC (Jump if Carry Flag Set), 2-18
JE (Jump if Equal), 2-19
JGE (Jump if Signed Greater Than or Equal), 2-19
JGT (Jump if Signed Greater Than), 1-19, 2-20
JH (Jump if Higher Unsigned), 1-19, 2-20
JLE (Jump if Less Than or Equal), 2-21
JLT (Jump if Signed Less Than), 2-21
JNC (Jump if Carry Flag is Clear), 2-22

J
JNE (Jump if Not Equal), 2-22
JNH (Jump if Not Higher Unsigned), 2-23
JNST (Jump if Sticky Bit is Clear), 2-23
JNV (Jump if Overflow Flag is Clear), 2-24
JNVT (Jump if Overflow Trap is Clear), 2-24
JST (Jump if Sticky Bit is Set), 2-25
Jump, 2-29, 2-50
  Conditional, 1-19
  if Carry Flag Clear, 2-22
  if Sticky Bit Clear, 2-23
  on Bit Clear, 2-17
  on Bit Set, 2-18
  on Carry Flag Set, 2-18
Jump to Self, 4-37
JV (Jump if Overflow Flag is Set), 2-25
JVT (Jump if Overflow Trap is Set), 2-26

L
LCALL (Long Call), 2-26
LD (Load Word), 2-27
LDB (Load Byte), 2-27
LDBSE (Load Integer with Short-Integer), 1-19, 2-28
LDBZE (Load Word with Byte), 2-28
Least Significant Bit—A/D Converter, 4-16
LJMP (Long Jump), 2-29
Load, 2-27
Load Sign Extended, 2-28
Load Zero Extended, 2-28
LOC0 LOC1 Program Lock Control, 1-13
Lock Modes, 1-13
Lock Program, 4-38
Long-Indexed Addressing, 1-16, 2-01
Long-Integer Operands, 1-15, 1-19
lreg, 2-01
LSB—Least Significant Bit—A/D Converter, 4-16

M
MA0 (Memory Address 0), 4-24
MA0–MA15 (Memory Address 0–15), 1-08
Memory
  Interface, 4-20
  Map, 1-04, 1-51
  Reserved Locations, 1-07
  Timings, 1-09
Memory Controller, 1-01, 1-02, 1-08
Missed Codes—A/D Converter, 4-15, 4-16
Mode 0 Timings Serial Port, 4-19
Mode 2 and 3 Timings Serial Port, 4-20
Modified Quick-Pulse Programming, 4-39
Monotonic—A/D Converter, 4-15, 4-16
MUL (Multiply Integers), 2-29, 2-30
MULB (Multiply Short Integers), 2-30, 2-31
Multiply, 2-29, 2-30, 2-31, 2-32, 2-33
Multiprocessor Communications, 1-39
MULU (Multiply Words), 2-31, 2-32
MULUB (Multiply Bytes), 2-32, 2-33

N

N Flag (see Negative Flag), 1-18
NEG (Negate Integer), 2-33
Negate Instruction, 2-33, 2-34
Negative Flag, 1-18
NEGB (Negate Short-Integer), 2-34
NMI (Non-Maskable Interrupt), 1-04, 4-01
No Missed Codes—A/D Converter, 4-15, 4-16
Noise Protection, 4-28
Non-Linearity—A/D Converter, 4-15, 4-16
Nonmaskable Interrupt, 1-04
NOP (No-Operation), 2-34
NORML (Normalize Long-Integer), 1-19, 1-27, 2-35
NOT (Complement Word), 2-35
NOTB (Complement Byte), 2-36

O

Off-Isolation—A/D Converter, 4-15, 4-17
Opcode, 2-01
Opcode List, 1-54, 1-55, 1-56
Open Drain Ports, 4-08
Operand Addressing, 1-15
Operand Types, 1-14
Operating Modes, 1-09
OR (Logical OR Words), 2-36
ORB (Logical OR Bytes), 2-37
Oscillator, 1-03, 4-01, 4-02
Output Ports, 1-40
Overflow Flag, 1-18, 2-24, 2-25
Overflow Trap, 1-18, 2-09, 2-24, 2-26

P

Packaging, 1-48, 4-29
Packaging Diagram, 1-49
PACT (Programming Active), 4-31
PALE (Programming ALE Input), 4-31
PC (Program Counter), 1-02, 2-01
PCCB (Programming CCB), 1-10, 4-30, 4-36
PCCR Programming Chip Configuration Register (PCCR), 4-30
PDO (Program Duration Overflow), 4-31
Phase
  Internal Clock, 1-03
Pin Description, 1-44, 1-45, 1-46
Pin List, 1-47
Pinouts, 1-50
PLM-96, 1-22
PLMREG, 1-22
PMODE (Programming Mode), 1-59, 4-30, 4-31
POP (POP Word), 2-37
Pop Flags, 2-38
Pop Word, 2-37
POPF (Pop Flags), 1-27, 2-38
Port 0, 1-40
  Timings, 4-19
Port 1, 1-40
  Timings, 4-19

P

Port 2, 1-39, 1-40, 1-59
  Alternate Functions, 1-39
  Timings, 4-19
Port 3, 1-40, 4-31
  Timings, 4-19
Port 4, 1-40, 4-31
  Timings, 4-19
Port Reconstruction, 4-28
Power Down, 1-05
Power Down Circuitry, 4-06
PROG (Programming Pulse), 4-31
Program Counter (PC), 1-02, 2-01
Program Lock, 1-13, 4-38
Program Status Word (PSW), 1-02, 1-17, 1-25
Program Verified, 4-31
Program/Verify Signals, 4-35
Programming, 4-30
  Auto, 4-32
  Configuration Byte, 4-36
  Gang-Auto, 4-32
  Gang-Slave, 4-35
  Mode Select, 4-31
  Modes, 4-30
  Run-Time, 4-37
  Slave, 4-34
Programming Active, 4-31
Programming ALE Input, 4-31
Programming Chip Configuration Byte (PCCB), 1-10, 4-30
Programming Duration Overflow, 4-31
Programming Mode Pin Definitions, 4-31
Programming Pulse, 4-31
PSW (Program Status Word), 1-02, 1-17, 1-25
Pulse Width Modulation (see PWM), 1-35
PUSH (Push Word), 2-38
Push Flags, 2-39
Push Word, 2-38
PUSHF (Push Flags), 1-27, 2-39
PVER (Program Verified Output), 4-31
PWM
  Using the HSO, 1-36
PWM (Pulse Width Modulation), 1-35, 4-09, 4-17

Q

Quasi-Bidirectional Hardware Connections, 4-06
Quasi-Bidirectional Port, 1-40, 4-06, 4-07
Queue, 1-08

R

RALU (Register/Arithmetic Logic Unit), 1-01, 1-02
RAM
  Internal Memory, 1-04
$\overline{RD}$ (Read), 1-09, 4-09, 4-20
Read, 1-09
Ready, 1-09, 1-13, 4-21
Ready Control, 1-13

R (Continued)
Receive Interrupt (RI), 1-37, 4-19
Register Direct Addressing, 1-15, 2-01
Register File, 1-01, 1-02, 1-04
Register Utilization, 1-22
REN (Receiver Enable), 1-37, 4-19
Repeatability—A/D Converter, 4-17
Reserved Memory Locations, 1-07
Reset, 1-03, 1-05, 1-43, 4-03
  Instruction, 2-40, 4-28
  Multiple Chip, 4-04
  Sequence, 4-03
  Status, 1-44
  Sync Mode, 1-44
Reset Signal, 1-43
Resolution—A/D Converter, 4-17
RET (Return), 2-39
Return, 2-39
RI (Receive Interrupt), 1-37, 4-19
ROM, 1-07
ROM/EPROM Dump Mode, 4-38
ROM/EPROM Lock, 1-13, 4-38
RST (Reset Instruction), 2-40, 4-28
Run-Time Programming, 4-37
RXD (Receive Pin), 4-09

S
SALE (Slave ALE), 4-31
Sample Delay—A/D Converter, 4-17
Sample Delay Uncertainty—A/D Converter, 4-17
Sample Time—A/D Converter, 4-17
Sample Time Uncertainty—A/D Converter, 4-17
Sample Window—A/D Converter, 4-17
Sampling Circuitry—A/D Input, 4-10
SBUF (Serial Port Buffer), 1-37
SCALL (Short Call), 2-40
Security Key, 4-38
Serial Port, 1-36
  Buffer (SBUF), 1-37
  Control/Status, 1-37
  Mode 0, 1-36
  Mode 0 Example, 4-20
  Mode 0 Timings, 4-19
  Mode 1, 1-37
  Mode 2, 1-37
  Mode 2 and 3 Timings, 4-20
  Mode 3, 1-37
  Timings, 4-19
SETC (Set Carry Flag), 2-41
SFR, 1-01, 1-04, 1-05
SFR Summary, 1-06, 1-57
Shift Indirect, 2-01
SHL (Shift Word Left), 2-41
SHLB (Shift Byte Left), 2-42
SHLL (Shift Double-Word Left), 2-43
Short-Indexed Addressing, 1-16, 2-01
Short-Integer Operands, 1-14, 1-19
SHR (Logical Shift Right Word), 2-44

S
SHRA (Arithmetic Shift Right Word), 2-45
SHRAB (Arithmetic Shift Right Byte), 2-46
SHRAL (Arithmetic Shift Right Double-Word), 2-47
SHRB (Logical Shift Right Byte), 2-48
SHRL (Logical Shift Right Double-Word), 2-49
SID (Slave ID), 4-31
Sign Extend, 2-15
Signature Word, 4-39
SIGND, 1-27
SJMP (Short Jump), 2-50
SKIP (Two Byte No-Operation), 2-50
Slave ALE, 4-31
Slave ID Number, 4-31
Slave PC, 1-02, 1-08
Slave Programming, 4-34
Slave Programming Mode Commands, 4-34
Slave Programming Pulse, 4-31
Software Overview, 1-14
Software Protection, 1-43
Software Standards, 1-22
Software Timers, 1-33
Software Trap, 2-55
Special Function Register (see SFR), 1-01
SPROG (Slave Programming Pulse), 4-31
SP_CON, 1-37, 1-38, 1-57
SP_STAT, 1-37, 1-38, 1-57
ST (Store Word), 2-51
ST Flag (see Sticky Bit), 1-18
Stack Pointer, 1-04
Stack Pointer Register Addressing, 1-17
Standard Bus Control, 1-11
Standard I/O Ports, 4-19
State Time, 1-03
State Time List, 1-54, 1-55, 1-56
Status and Control Registers, 1-41
STB (Store Byte), 2-51
Sticky Bit, 1-18, 2-25
Store, 2-51
SUB (Subtract Words), 2-52
SUBB (Subtract Bytes), 2-53
SUBC (Subtract Words with Borrow), 2-54
SUBCB (Subtract Bytes with Borrow), 2-54
Subroutine Linkage, 1-22
Subtract, 2-52, 2-53, 2-54
Successive Approximation—A/D Converter, 4-17
Sync Mode, 4-05
System Bus, 1-08, 1-40, 4-09, 4-20
System Bus Timings, 4-21
System Verification, 4-26

T
T2CLK (TIMER2 Clock), 4-19
T2RST (TIMER2 Reset), 1-32
TAVDV (ADDRESS Valid to DATA Valid), 4-22, 4-27
TAVGY (ADDRESS Valid to BUSWIDTH Valid), 4-22

T (Continued)
TAVLL (ADDRESS Valid to ALE/$\overline{\text{ADV}}$ Low), 4-23, 4-27
TB8/RB8, 1-37
TCHCH (CLKOUT High to CLKOUT High), 4-23
TCHCL (CLKOUT High to CLKOUT Low), 4-23
TCLLH (CLKOUT Low to ALE High), 4-23
TCLVL (CLKOUT Low to ALE/$\overline{\text{ADV}}$ Low), 4-23
TCLYX (READY Hold after CLKOUT Low), 4-22
Temperature Coefficients—A/D Converter, 4-17
Terminal Based Characteristic—A/D Converter, 4-14, 4-17
THLHH ($\overline{\text{WRL}}$ $\overline{\text{WRH}}$ low to $\overline{\text{WRL}}$ $\overline{\text{WRH}}$ high), 4-23, 4-27
TI (Transmit Interrupt), 1-37, 4-19
Timer 2, 1-32
Timer Interrupts, 1-28
Timings
  Definitions, 4-22, 4-23
  HSI/HSO, 4-18
  I/O, 4-18
  I/O Ports, 4-19
  Internal, 1-03, 4-03
  Serial Port, 4-19
  System Bus, 4-21
TLHLL (ALE/$\overline{\text{ADV}}$ High to ALE/$\overline{\text{ADV}}$ Low), 4-23
TLLAX (ALE/$\overline{\text{ADV}}$ Low to ADDRESS Invalid), 4-23
TLLCH (ALE/$\overline{\text{ADV}}$ Low to CLKOUT High), 4-23
TLLGV (ALE/$\overline{\text{ADV}}$ Low to BUSWIDTH Valid), 4-22
TLLGX (BUSWIDTH Hold after ALE/$\overline{\text{ADV}}$ Low), 4-22
TLLHL (ALE/$\overline{\text{ADV}}$ Low to $\overline{\text{WRL}}$ $\overline{\text{WRH}}$ Low), 4-23
TLLRL (ALE/$\overline{\text{ADV}}$ Low to $\overline{\text{RD}}$ Low), 4-23
TLLRL (ALE/$\overline{\text{ADV}}$ Low to RD or $\overline{\text{WR}}$ Low), 4-27
TLLYH (ALE/$\overline{\text{ADV}}$ Low to READY High), 4-22
TLLYL (ALE/$\overline{\text{ADV}}$ Low to READY Low), 4-22
TOHCH (XTAL1 High to CLKOUT High), 4-23
TOSC (Oscillator Period), 4-22
TQVHL (OUTPUT Valid to $\overline{\text{WRL}}$ $\overline{\text{WRH}}$ Low), 4-23
TQVWH (OUTPUT Valid to $\overline{\text{WR}}$ High), 4-23, 4-28
Transmit Interrupt (TI), 1-37, 4-19
TRAP (Software Trap), 1-27, 2-55
TRHBX ($\overline{\text{RD}}$ High to INST $\overline{\text{BHE}}$ AD8–AD15 Invalid), 4-23
TRHDZ ($\overline{\text{RD}}$ High to DATA Float), 4-22
TRHLH ($\overline{\text{RD}}$ High to ALE/$\overline{\text{ADV}}$ High), 4-23
TRLDV ($\overline{\text{RD}}$ Low to DATA Valid), 4-22, 4-27
TRLRH ($\overline{\text{RD}}$ Low to $\overline{\text{RD}}$ High), 4-23
TWHBX ($\overline{\text{WR}}$ High to INST $\overline{\text{BHE}}$ AD8–AD15 Invalid), 4-23
TWHLH (Write High to ALE/$\overline{\text{ADV}}$ High), 4-23
TWHQX ($\overline{\text{WR}}$ High to OUTPUT Not valid), 4-23
TWLWH ($\overline{\text{WR}}$ Low to $\overline{\text{WR}}$ High), 4-23
TXD (Transmit Pin), 4-09
TYLYH (READY Low to READY High), 4-22

V
V Flag (see Overflow Flag), 1-18
V_{CC}, 4-01
V_{CC} Rejection—A/D Converter, 4-15, 4-17
V_{PD} (Powerdown Voltage), 1-05, 4-01
V_{REF} (Analog Voltage Reference), 1-33, 4-01, 4-11
V_{SS}, 4-01
V_{SS1}, 4-01
V_{SS2}, 4-01
VT Flag (see Overflow Trap), 1-18

W
waop, 2-01
Watchdog Timer, 1-43, 4-05
  Disabling, 1-43, 4-05
Word Dump Command, 4-35
Word Operands, 1-14, 1-19
$\overline{\text{WR}}$ (Write), 1-09, 1-11, 4-09, 4-20, 4-24
wreg, 2-01
$\overline{\text{WRH}}$ (Write High), 1-11, 1-12, 4-20, 4-24
Write Strobe Mode, 1-12
$\overline{\text{WRL}}$ (Write Low), 1-11, 1-12, 4-20, 4-24

X
XOR (Logical Exclusive-or Words), 2-55
XORB (Logical Exclusive-or Bytes), 2-56
XTAL Inputs, 1-03, 4-01, 4-02

Z
Z Flag (see Zero Flag), 1-18
Zero Flag, 1-18
Zero Offset—A/D Converter, 4-15, 4-17
Zero Register Addressing, 1-17